# TASKGROUND: Structured Executable Task Inference for Full-Scene Household Reasoning

**ZhiYuan Feng**[1,†] **Yu Deng**[2] **Ruichuan An**[3,†] **Zhenhua Liu**[2] **Qixiu Li**[1,†]
**Keming Wu**[1] **Zhiying Du**[4,†] **Weijie Wang**[5] **Haoxiao Wang**[5] **Shuang Chen**[5]
**Sicheng Xu**[2] **Yaobo Liang**[2] **Jiaolong Yang**[2,✉] **Baining Guo**[2]
[1]Tsinghua University [2]Microsoft Research Asia [3]Peking University
[4]Fudan University [5]Zhejiang University

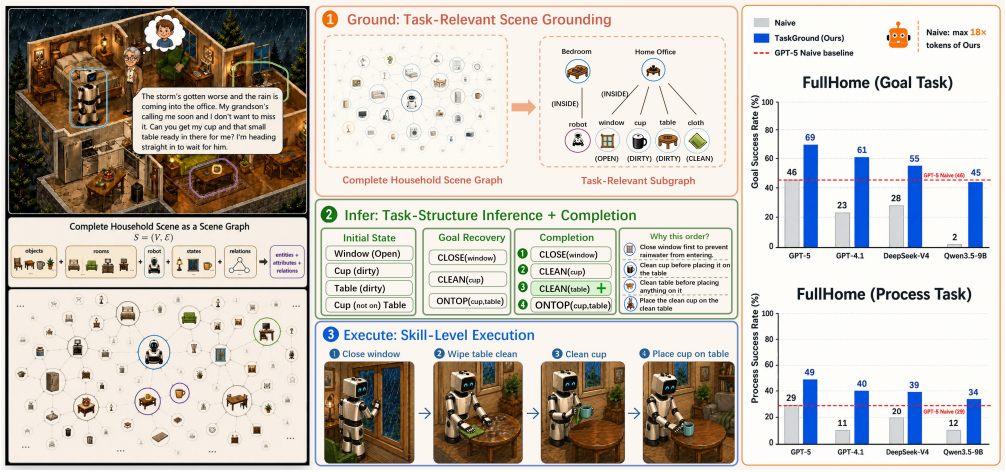



Figure 1: Overview of **TaskGround** for full-scene household reasoning. It grounds complete scenes, infers executable task structure, and compiles grounded skill-level actions, improving task success rates on our FullHome benchmark with up to 18× lower total input-token cost.

## Abstract

In real home deployments, household agents must often operate from a complete household scene and a situated household request, rather than from a clean task specification. Such requests require agents to identify task-relevant entities, recover intended task conditions, and resolve ordering constraints from the surrounding scene context. We formalize this capability as **full-scene household reasoning**: given a complete structured scene graph and a situated household request, an agent must infer executable task structure before producing a grounded skill-level action sequence. This setting is challenging because complete household scenes contain substantial task-irrelevant information, making direct complete-scene prompting inefficient and error-prone. In practical deployment, this challenge is further amplified by privacy and local compute constraints, which favor compact open-weight models with limited long-context reasoning ability. We propose **TaskGround**, a training-free and model-agnostic *Ground–Infer–Execute* framework that grounds complete scenes into compact task-relevant scene slices while preserving instance identity, infers executable task structure, completes it using fixed household priors, and compiles it into grounded skill-level action sequences. To evaluate this

[†]Work done during a research internship at Microsoft Research Asia.
[✉]Corresponding author.

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

setting, we introduce **FullHome**, a simulator-backed, human-validated evaluation suite of 400 household tasks spanning diverse home-scale environments and both goal-oriented and process-constrained requirements. On FullHome, TaskGround improves task success rates by large margins across both proprietary and open-weight models. Notably, it makes Qwen3.5-9B competitive with GPT-5 under direct complete-scene prompting while reducing total input-token cost by up to 18×. Our results identify executable task-structure inference as a central bottleneck in full-scene household reasoning and show that structured grounding can make compact local models substantially more effective for task-level household reasoning.

## 1 Introduction

In real home deployments, household requests are often embedded in everyday situations rather than written as clean task specifications. Consider the example in Fig. 1: a grandmother notices rain blowing into her home office while waiting for a call from her grandson. She says to her household robot, *"The storm is getting worse, and rain is coming into the office. My grandson is calling soon. Can you get my cup and the small table ready in there?"* Although natural, this request is far from a clean task specification. It does not explicitly identify all task-relevant entities, such as the office window, the dirty cup, the table, and the target room. Nor does it specify the intended task conditions or process requirements. Given the scene context, however, the robot should infer that it needs to close the window to prevent further rain from entering, clean the table before placing the cup on it, clean the dirty cup, and finally place the cup on the table. Thus, before generating actions, the agent must infer an intermediate representation from the complete scene and the context-dependent request. We refer to this representation as *executable task structure*: the task-relevant entities, target conditions, relations, and process requirements sufficient for producing executable actions. This example motivates **full-scene household reasoning**. Given a *complete household scene* and a *situated household request*, an agent must infer executable task structure and produce a grounded skill-level action sequence. Here, a complete household scene is represented as a structured scene graph containing the full scene context available to the agent, including both task-relevant and irrelevant objects, states, and relations. A situated household request is a naturally phrased, context-dependent request whose intended meaning is grounded in the current household scene rather than fully specified by language alone. The output is a grounded skill-level action sequence, such as walking to the window, closing it, cleaning the table, cleaning the cup, and placing the cup on the table.

Recent progress in embodied planning with large language models (LLMs) has shown that LLMs can support high-level household decision making, including action-sequence and programmatic plan generation [13, 36, 37, 45, 46], skill or policy grounding [1, 22], and interactive or personalized assistance [14, 44]. However, these systems typically study a different information condition from full-scene household reasoning. They often assume that the user instruction, task goal, relevant objects, or executable constraints have already been stated, structured, or narrowed before planning begins. As a result, the main challenge is often to map a given task description to executable actions or skills. In contrast, our setting starts from a *complete household scene* and a *situated household request*, and requires the agent to produce a grounded skill-level action sequence. To succeed, the agent must first infer *executable task structure*, which specifies the relevant entities, target conditions, and process requirements needed to generate executable actions. Thus, full-scene household reasoning evaluates whether an agent can infer what to plan for from complete scene context, rather than only whether it can execute a task whose structure has already been exposed.

This information condition is especially important for practical home deployment. A complete household scene may contain many rooms, objects, states, affordances, and relations, making direct prompting over complete scenes costly and error-prone. Detailed household states can also reveal private information about personal spaces, daily routines, and object usage patterns, making it undesirable to rely solely on frontier proprietary models for complete-scene inputs [4, 10, 27, 43]. Meanwhile, real homes vary widely in layout, object distribution, routines, and user preferences, so practical systems should adapt to new households without task-specific training or repeated model tuning. Local or on-device deployment further imposes memory, compute, energy, and latency constraints, favoring compact open-weight models that are easier to deploy but less capable of reasoning over long, cluttered inputs [29, 41, 49]. These challenges motivate a low-cost, model-

agnostic mechanism for inferring executable task structure from complete household scenes and situated household requests under limited model capacity.

Motivated by this need, we propose **TaskGround**, a training-free and model-agnostic *Ground–Infer–Execute* framework for full-scene household reasoning. As illustrated in Fig. 1, TaskGround avoids directly generating a grounded skill-level action sequence from a long and cluttered complete household scene. Instead, given a complete household scene and a situated household request, it first grounds the scene into a compact task-relevant scene slice by querying the language model over scene entities and reconstructing the needed local context around selected entities, substantially reducing the input cost of downstream reasoning. Over this grounded context, TaskGround infers *executable task structure*, specifying the task-relevant entities, target conditions, and ordering requirements needed for execution. Because situated household requests often leave goals or intermediate steps implicit, TaskGround further completes the inferred structure with non-oracle household priors and process-critical subgoals, reflecting generic household patterns or pre-specified routines without using ground-truth task structures or evaluation outcomes. Finally, TaskGround compiles the completed structure into a grounded skill-level action sequence through a task-agnostic skill interface, such as walking to an object, closing a window, cleaning a table, and placing a cup on it. This design preserves scene-instance identity across stages and represents task structure as ordered goal predicates, rather than passing a free-form plan between modules. This decomposition reduces the reasoning burden on the base model while preserving executability, making TaskGround efficient, compatible with both proprietary and open-weight models, and suitable for deployment without task-specific training.

To evaluate this problem systematically, we introduce **FullHome**, a simulator-backed evaluation suite of 400 carefully designed and human-validated household tasks. Using complete structured scene states and predefined skill interfaces, FullHome isolates task-level reasoning from perception and low-level control. Unlike many prior household environments and embodied-agent interfaces that often expose programmatic task specifications, explicit goals, curated objects, or structured constraints before planning [5, 19, 20, 28, 38], FullHome evaluates agents under the full-scene information condition: given a complete household scene and a situated household request, the agent must produce a grounded skill-level action sequence without being given task-relevant objects, target states, or process requirements. We conduct comprehensive experiments on FullHome across proprietary and open-weight models. TaskGround consistently improves success rates across model families. Notably, it makes Qwen3.5-9B competitive with GPT-5 under direct complete-scene prompting, while reducing total input-token cost by up to $18\times$. Ablation studies further confirm that scene grounding, task-structure inference, task-structure completion, and skill-level execution each contribute to the final performance. Together, these results demonstrate that structured grounding can make compact local models substantially more effective for full-scene household reasoning without task-specific training.

In summary, our main contributions are as follows:

- We formalize **full-scene household reasoning** as a task-level reasoning problem for household agents: given a complete household scene and a situated household request, the agent must produce a grounded skill-level action sequence without being given task-relevant objects, target states, or process requirements.
- We propose **TaskGround**, a training-free and model-agnostic *Ground–Infer–Execute* framework for this setting. TaskGround grounds complete household scenes into compact task-relevant scene slices, infers and completes executable task structure, and compiles the completed structure into grounded skill-level action sequences.
- We introduce **FullHome**, a simulator-backed evaluation suite for systematically measuring full-scene household reasoning. FullHome contains 400 carefully designed and human-validated household tasks across diverse home-scale environments, covering both goal-oriented and process-constrained requirements.
- We conduct comprehensive experiments across proprietary and open-weight models. TaskGround consistently improves success rates, makes Qwen3.5-9B competitive with GPT-5 under direct complete-scene prompting, and reduces total input-token cost by up to $18\times$ on larger complete scenes.

## 2 Problem Formulation

We formalize **full-scene household reasoning** as the task of mapping a *complete household scene* and a *situated household request* to a grounded skill-level action sequence. A complete household scene is represented as a scene graph $\mathcal{S} = (\mathcal{V}, \mathcal{E})$, where nodes in $\mathcal{V}$ denote household entities, such as rooms, objects, surfaces, containers and the agent, and edges in $\mathcal{E}$ denote relations among them, such as containment, support, proximity, and room membership. Each entity may also carry attributes describing its category, current state, and available affordances. The scene is *complete* in the sense that it contains the full household context available to the agent, including both task-relevant and irrelevant entities. We assume complete, accurate scene states and no unexpected external changes during execution.

An instance of full-scene household reasoning consists of an initial complete household scene $\mathcal{S}_0$ and a situated household request $u$. A situated household request is a naturally phrased, context-dependent user request whose intended meaning is grounded in the current household scene rather than fully specified by language alone. The agent takes $(\mathcal{S}_0, u)$ as input and outputs a grounded skill-level action sequence:

$$(\mathcal{S}_0, u) \xrightarrow{\text{agent}} a_{1:T} = (a_1, a_2, \ldots, a_T).$$

Each action $a_t$ is an executable skill-level command with grounded entity arguments, such as `WalkTo(window)`, `Close(window)`, `Wipe(table)`, `Clean(cup)`, or `Place(cup, table)`. Object names in these examples abbreviate unique scene-instance identifiers. These skills abstract low-level control into object-centric actions that a household agent is assumed to be able to perform, and each action must refer to entities in the current scene. Executing a valid action updates the scene state, written as $\mathcal{S}_t = \mathrm{Step}(\mathcal{S}_{t-1}, a_t)$, where $\mathrm{Step}$ denotes the environment transition induced by the executed action. Invalid actions, such as placing an object without holding it or putting an object into a closed container, lead to execution failure.

Although the required output is only $a_{1:T}$, the input pair $(\mathcal{S}_0, u)$ implicitly defines what the agent should accomplish. We describe this hidden task semantics as

$$\tau = (\mathcal{G}, \mathcal{P}),$$

where $\mathcal{G}$ is a set of goal conditions and $\mathcal{P}$ is a set of process constraints. Each goal condition $g \in \mathcal{G}$ describes a desired state or relation that should hold after task completion. For example, in the scenario in Fig. 1, the request may imply goals such as `CLOSED(window)`, `CLEAN(table)`, `CLEAN(cup)`, and `ON(cup, table)`. These goals specify the intended outcome, even though the user does not explicitly list them in the request.

Some household requests also imply temporal or process requirements. We represent these requirements with constraints in $\mathcal{P}$, written in the form $g_i$ `then` $g_j$, meaning that condition $g_i$ should be achieved before condition $g_j$. For example, the agent should close the window before continuing to prepare the office area, and it should clean the table and cup before placing the cup on the table. This can be expressed with constraints such as `CLOSED(window) then CLEAN(table)`, `CLEAN(table) then ON(cup, table)`, and `CLEAN(cup) then ON(cup, table)`. Multiple constraints can be combined by conjunction. These constraints need not impose a total order on all actions. Such process constraints capture requirements that are not determined by the final state alone.

Importantly, $\tau = (\mathcal{G}, \mathcal{P})$ is not given to the agent and is not required as the output. This evaluation specification is distinct from TaskGround's inferred structure $\tilde{\tau}$, introduced in Sec. 3. The agent only receives the complete household scene and the situated household request, and must produce an executable grounded skill-level action sequence. Full-scene household reasoning therefore tests whether an agent can infer the task-relevant entities, intended goals, and process requirements from complete scene context and situated language, rather than merely execute a task whose objects, goals, and constraints have already been specified.

## 3 TaskGround: Ground–Infer–Execute Framework

The formulation above highlights three coupled challenges for deployable full-scene household reasoning. First, the complete household scene can contain many irrelevant entities and relations, making direct complete-scene prompting costly and error-prone. Second, the intended goals and

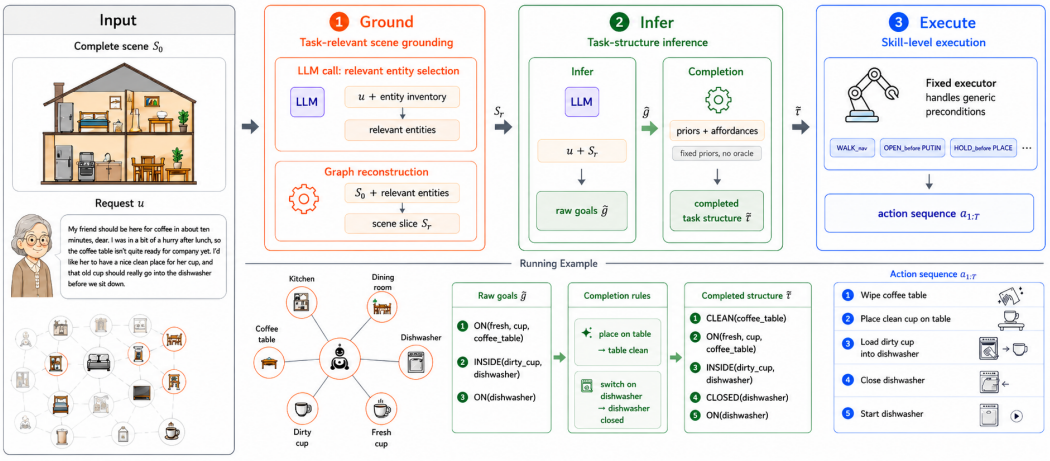



Figure 2: Overview of **TaskGround**. Given a complete household scene and a situated household request, TaskGround follows a *Ground–Infer–Execute* decomposition. The orange module grounds the input into a compact task-relevant scene slice; the green module infers and completes executable task structure; the blue module compiles the completed structure into a grounded skill-level action sequence. The bottom row shows a running example from scene grounding to completed task structure and executable actions.

process requirements are often implicit in the situated household request, so the agent must infer what task structure should be achieved before acting. Third, the final output must be executable: actions must be grounded to available entities and satisfy skill preconditions, otherwise the execution may fail even when the high-level intent is correct. Practical deployment further requires this capability without task-specific training whenever the home, user, or base model changes. To address these challenges, we propose **TaskGround**, a training-free and model-agnostic *Ground–Infer–Execute* framework.

As illustrated in Fig. 2, TaskGround aligns these challenges with three stages. Given an initial complete household scene $\mathcal{S}_0$ and a situated household request $u$, it first grounds the scene into a compact task-relevant scene slice, then infers and completes executable task structure, and finally compiles the completed structure into grounded skill-level actions:

$$\text{Ground: } \mathcal{S}_r = R(\mathcal{S}_0, u), \quad \text{Infer: } \hat{\mathbf{g}} = T(\mathcal{S}_r, u),\ \tilde{\tau} = C(\hat{\mathbf{g}}), \quad \text{Execute: } a_{1:T} = E(\tilde{\tau}, \mathcal{S}_r). \tag{1}$$

Here, $R$, $T$, $C$, and $E$ denote the scene grounder, task-structure inference module, completion module, and executor, respectively. $R$ and $T$ each use one LLM call; $C$ and $E$ are deterministic. The completed structure $\tilde{\tau}$ is an ordered goal sequence used for execution, and TaskGround does not access the hidden task semantics $\tau = (\mathcal{G}, \mathcal{P})$ defined in Sec. 2.

**Ground: Task-Relevant Scene Grounding.** The grounding stage produces a compact task-relevant scene slice $\mathcal{S}_r$ from the complete household scene $\mathcal{S}_0$. TaskGround first extracts an entity inventory from $\mathcal{S}_0$, including object and room names with lightweight state descriptions, and provides this inventory together with the situated request $u$ to the language model. The model selects room–object-class pairs, deterministically resolved to all matching nodes $\hat{\mathcal{V}}_r$, which may be explicitly mentioned in the request or implicitly required by the task.

TaskGround then reconstructs the grounded scene slice by expanding $\hat{\mathcal{V}}_r$ over the original scene graph. Specifically, it adds the containing rooms, the agent node and state, and execution-relevant neighboring entities when needed, and then restores all relations in $\mathcal{S}_0$ whose endpoints are included in the expanded node set. Formally, if $\mathcal{V}_r$ is the expanded node set, the grounded slice is

$$\mathcal{S}_r = (\mathcal{V}_r, \mathcal{E}_r), \qquad \mathcal{E}_r = \{(v_i, v_j) \in \mathcal{E}_0 \mid v_i, v_j \in \mathcal{V}_r\}.$$

This node-first grounding removes irrelevant clutter while preserving the local context needed for downstream task-structure inference and execution. As reported in Sec. 5, this reduces total input-token cost by up to 18× compared with direct complete-scene prompting.

**Infer: Task-Structure Inference and Completion.** Given the grounded scene slice $\mathcal{S}_r$ and request $u$, the inference module $T$ predicts an ordered goal sequence $\hat{\mathbf{g}} = (\hat{g}_1, \ldots, \hat{g}_M)$. Each goal atom describes a desired state or relation over grounded entities, such as `ON(fresh_cup, coffee_table)` or `INSIDE(dirty_cup, dishwasher)`. This goal-level output focuses the language model on task interpretation before skill-level action generation.

The initial sequence may still miss process-critical goals that are implicit in household execution. For example, as shown in Fig. 2, the model may predict `INSIDE(dirty_cup, dishwasher)` and `ON(dishwasher)`, but omit `CLOSED(dishwasher)`, which is required before starting the dishwasher. The completion module $C$ therefore augments $\hat{\mathbf{g}}$ using fixed household priors and object affordances, such as closing an appliance before turning it on or cleaning a surface before placing objects on it. These priors are specified before evaluation and do not access hidden goals, oracle actions, task identifiers, or evaluation outcomes. Scene affordances are shared by all methods; $C$'s dependence on them and the fixed prior library is implicit. The result is a completed executable task structure $\tilde{\tau} = (\tilde{g}_1, \ldots, \tilde{g}_K)$, an ordered goal sequence that contains both final goals and process-critical intermediate goals.

**Execute: Skill-Level Execution.** The execution stage converts the completed executable task structure $\tilde{\tau}$ into a grounded skill-level action sequence $a_{1:T}$. Given $\tilde{\tau}$ and the grounded scene slice $\mathcal{S}_r$, the executor $E$ realizes each ordered goal atom as a scene-state change using the agent's available skills, such as navigation, manipulation, cleaning, container operation, or appliance control. Execution preserves goal order and updates the symbolic state after each action. In Fig. 2, this corresponds to the blue-panel action sequence. The executor also handles generic preconditions required for executable actions, such as navigating near an object before manipulation, holding an object before placing it, or opening a container before insertion. It does not infer task-relevant entities, implicit goals, or process requirements; it only realizes the completed structure produced by the previous stages. This delegates action realization to a fixed skill-level executor, reducing unsupported or hallucinated actions while preserving executability.

## 4 FullHome Evaluation Protocol

We instantiate full-scene household reasoning with **FullHome**, a simulator-backed evaluation suite of 400 carefully designed and human-validated household tasks built on executable household environments, including VirtualHome [28] and BEHAVIOR [38]. FullHome changes the test-time information condition of existing simulator-backed household evaluations: the agent receives only the complete household scene $\mathcal{S}_0$ and the situated household request $u$, and must output a grounded skill-level action sequence $a_{1:T}$. It is not given task-relevant objects, candidate action arguments, explicit goals, target states, reference action sequences, or process constraints. The predicted sequence is symbolically executed on simulator-derived scene states, producing a trajectory $\rho = (\mathcal{S}_0, a_1, \mathcal{S}_1, \ldots, a_T, \mathcal{S}_T)$, which is evaluated against the hidden task structure $\tau = (\mathcal{G}, \mathcal{P})$.

FullHome contains two complementary task families. We use $\models$ to denote satisfaction. For goal-oriented tasks, success requires the final scene state to satisfy all goal conditions, i.e., $\mathcal{S}_T \models \mathcal{G}$. For process-constrained tasks, success additionally requires the execution trajectory to satisfy the process constraints, i.e., $\mathcal{S}_T \models \mathcal{G}$ and $\rho \models \mathcal{P}$. We report full-split success rates, counting parsing and execution failures as unsuccessful tasks. This protocol separates failures in reaching the desired final state from failures in satisfying required intermediate steps or ordering constraints. It therefore evaluates whether agents can infer executable task structure from complete household scenes and situated household requests, rather than relying on task structure exposed before planning. Additional benchmark construction and validation details are provided in Appendix B.

## 5 Experiments

We study whether TaskGround improves full-scene household reasoning, which components drive the gains, and whether structured grounding helps compact open-weight models approach frontier direct prompting with lower input cost.

Table 1: Main results on FullHome. **Goal SR** and **Process SR** are success rates on goal and process tasks. Cells shaded exceed *GPT-5 Naive*, the frontier direct-prompting reference for each column. (+Δ) denote absolute gains of *TaskGround* over the same model's *Naive* result. **Bold** marks the best result within each model group.

| Model | Method | VirtualHome | | BEHAVIOR | |
|---|---|---|---|---|---|
| | | Goal SR | Process SR | Goal SR | Process SR |
| **Proprietary models** | | | | | |
| GPT-4o | Naive | 8.0 | 5.0 | 32.3 | 23.3 |
| | TaskGround | 49.0 (+41.0) | 37.0 (+32.0) | 52.3 (+20.0) | 35.5 (+12.2) |
| GPT-4.1 | Naive | 14.0 | 4.0 | 32.3 | 17.1 |
| | TaskGround | 57.0 (+43.0) | 34.0 (+30.0) | **64.6** (+32.3) | 45.7 (+28.6) |
| Gemini-2.5-Flash | Naive | 0.5 | 2.0 | 9.2 | 5.7 |
| | TaskGround | 48.5 (+48.0) | 19.0 (+17.0) | 38.5 (+29.3) | 37.1 (+31.4) |
| GPT-5 | Naive | 45.5 | 26.0 | 46.2 | 31.4 |
| | TaskGround | **73.5** (+28.0) | **46.0** (+20.0) | **64.6** (+18.4) | **51.4** (+20.0) |
| **Open-weight models** | | | | | |
| DeepSeek-V4-Flash | Naive | 28.5 | 15.0 | 27.7 | 25.7 |
| | TaskGround | **63.0** (+34.5) | **36.0** (+21.0) | **47.7** (+20.0) | **42.9** (+17.2) |
| MiMo-V2-Flash | Naive | 5.0 | 1.0 | 10.8 | 11.4 |
| | TaskGround | 43.5 (+38.5) | 23.0 (+22.0) | 43.1 (+32.3) | 40.0 (+28.6) |
| Gemma-3-12B | Naive | 0.0 | 1.0 | 0.0 | 5.7 |
| | TaskGround | 37.0 (+37.0) | 19.0 (+18.0) | 29.2 (+29.2) | 34.3 (+28.6) |
| Qwen3.5-9B | Naive | 0.0 | 10.0 | 1.5 | 14.3 |
| | TaskGround | 47.5 (+47.5) | 29.0 (+19.0) | 43.1 (+41.6) | 40.0 (+25.7) |

### 5.1 Experimental Setup

FullHome contains 400 tasks. For each task, the agent receives only $(\mathcal{S}_0, u)$ and must output a grounded skill-level action sequence, without access to explicit task-relevant objects, goals, constraints, or reference action sequences. We evaluate proprietary and open-weight models, including GPT-4o [15], GPT-4.1 [24], Gemini-2.5-Flash [8], GPT-5 [25], DeepSeek-V4-Flash [9], MiMo-V2-Flash [47], Gemma-3-12B-it [39], and Qwen3.5-9B [30], covering both frontier proprietary systems and compact models suitable for local deployment. We report **Goal SR** and **Process SR**: Goal SR measures whether the final executed state satisfies the goal conditions, while Process SR additionally requires the execution trajectory to satisfy process constraints.

### 5.2 Main Results

We compare **Naive** and **TaskGround**. Naive directly prompts the model with the complete household scene and situated household request to generate grounded skill-level actions. Table 1 reports the results. TaskGround consistently improves over Naive across all models, environments, and task types, showing that direct complete-scene action generation is brittle under situated household requests, while structured grounding and executable task-structure inference provide robust gains. TaskGround also narrows the gap between compact open-weight models and frontier direct prompting: Qwen3.5-9B with TaskGround becomes competitive with GPT-5 under Naive prompting, matching or exceeding it in three of the four environment–metric comparisons. Moreover, TaskGround improves GPT-5 itself, suggesting that the gains come from a better problem decomposition rather than only compensating for weaker base models.

### 5.3 Framework Ablation

We conduct framework ablations on the VirtualHome split of FullHome, as shown in Fig. 3. VirtualHome uses a single underlying household environment while varying the task-specific scene states, making it suitable for controlled component analysis and closer to a known-home deployment setting,

Table 2: Grounding and task-structure recovery diagnostics on the VirtualHome split of FullHome. **Recall / Prec.** measure scene-entity coverage, and **Task Rec.** measures recovery of hidden task conditions. **bold** marks the best Task Rec. within each model class; highlights the Ground+Act process-task regression despite high recall; uncolored **bold** marks the best Recall / Prec. entry.

| Model | Method | Goal Task | | | Process Task | | |
|---|---|---|---|---|---|---|---|
| | | Recall | Prec. | Task Rec. | Recall | Prec. | Task Rec. |
| GPT-5 | Naive | 80.8 | **75.5** | 77.9 | 78.0 | **86.5** | 49.8 |
| | Retrieve+Act | 88.8 | 40.1 | 76.3 | 83.9 | 56.3 | 51.6 |
| | Ground+Act | **96.0** | 32.4 | 81.8 | 90.0 | 47.4 | 49.3 |
| | TaskGround | **96.0** | 31.7 | **91.2** | **91.7** | 47.5 | **67.2** |
| Qwen3.5-9B | Naive | 37.4 | **43.2** | 18.0 | 27.1 | 36.9 | 21.3 |
| | Retrieve+Act | 50.5 | 22.7 | 19.5 | 44.6 | 35.5 | 25.0 |
| | Ground+Act | **86.5** | 35.8 | 43.8 | **80.7** | **50.3** | 15.8 |
| | TaskGround | 85.6 | 37.3 | **79.3** | 80.5 | 49.7 | **52.5** |

where house-specific priors or routines can be specified before evaluation. We compare five variants. **Naive** directly generates skill-level actions from the complete scene and request. **Ground-Act** first applies scene grounding, but still asks the model to directly generate actions from the grounded scene slice. **Ground-Infer-Execute w/o Completion** removes the completion module from TaskGround. **TaskGround** uses the full framework. Finally, **GT Scene Slice-Infer-Execute** replaces the predicted scene slice with a task-relevant scene slice to estimate how much performance can improve if grounding is near-perfect. The variants also differ in input cost: **Naive** serializes the complete household scene and uses about $18\times$ total input tokens, while the grounded-slice variants operate on compact task-relevant scene slices and are normalized to $1\times$ in Fig. 3.

The full framework consistently outperforms its non-oracle ablations, while grounding alone does not always improve success. Grounding reduces irrelevant scene clutter, but Ground-Act remains weak for several models, indicating that a smaller input alone does not solve the task. Adding task-structure inference gives large gains, especially for GPT-4o, GPT-4.1, and Qwen3.5-9B, suggesting that these models struggle to directly turn a scene and request into executable skill-level actions. Completion further improves performance by recovering missing goals and process-critical steps, such as closing an appliance before starting it or cleaning a surface before placing objects on it. The GT-scene-slice diagnostic remains above TaskGround, showing that scene grounding is still a remaining bottleneck; meanwhile, its performance is still not perfect, indicating that downstream task-structure inference and execution remain challenging even with the relevant scene context.

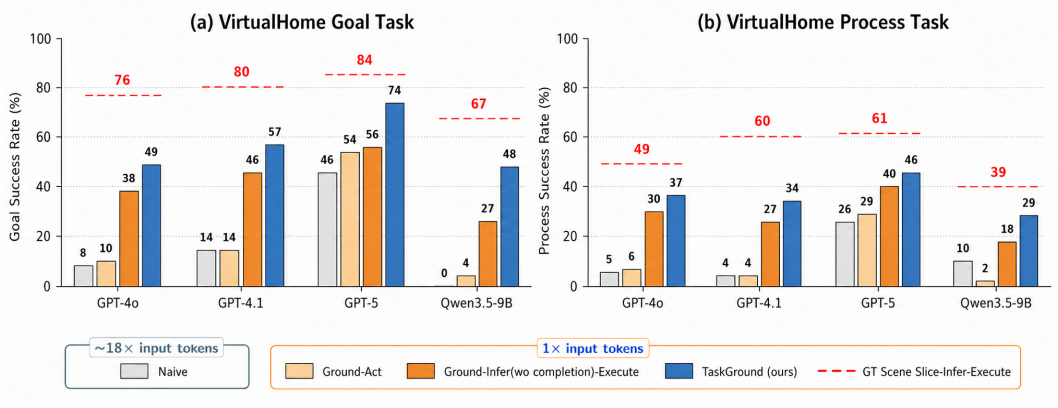



Figure 3: Framework ablation on the VirtualHome split of FullHome. Bars report **Goal SR** on goal-oriented tasks and **Process SR** on process-constrained tasks. The red dashed line shows a diagnostic GT-scene-slice setting that replaces the predicted scene slice with the task-relevant scene slice to isolate the grounding bottleneck.

### 5.4 High Grounding Recall Is Not Sufficient

Table 2 analyzes whether scene grounding recovers the context needed for task-structure recovery. **Retrieve+Act** directly asks the model to select relevant object IDs before action generation, while **Ground+Act** uses TaskGround's grounded scene slice but still directly generates actions. **Task Rec.** measures the fraction of recovered hidden task conditions: goal predicates for goal tasks, and both goal and process conditions for process tasks. The results show that high grounding recall alone does not ensure task-condition recovery. Naive prompting often has higher precision but much lower recall, especially for Qwen3.5-9B, indicating that it focuses on obvious entities while missing task-relevant context. TaskGround substantially improves recall, but Ground+Act still lags behind TaskGround in Task Rec. The blue-highlighted Qwen3.5-9B process result illustrates this gap: Ground+Act has higher reported Recall than Retrieve+Act, but lower Task Rec. This suggests that more relevant scene context alone does not ensure successful task-structure recovery. TaskGround addresses this by converting the grounded scene slice into executable task structure before skill-level execution, substantially improving Task Rec.

## 6 Related Work

### 6.1 Language Models for Household Embodied Agents

Large language models have been widely used as high-level decision-making modules for embodied agents. Prior work has explored LLMs for generating action sequences, programs, or hierarchical plans from natural-language instructions [13, 31, 36, 37, 45, 46], grounding high-level decisions in executable skills, affordances, robot policies, or motion-planning feedback [1, 22, 42], and supporting interactive, uncertainty-aware, or personalized assistance [14, 32, 44]. These works demonstrate the promise of LLMs for embodied decision making, but generally operate on stated instructions, goals, task descriptions, or bounded contexts where the task intent is explicit or can be clarified within the given interface. More closely related are methods that combine LLM planning with structured retrieval, symbolic intermediates, or graph-based planning: EMBODIED RAG retrieves task-relevant subgraphs for LLM-based planning [3]; UniPlan and DC-SGG use symbolic scene parsing and PDDL-style planning interfaces [12, 48]; BrainBody-LLM, LookPlanGraph, and SG2 improve embodied planning through feedback, simulator checks, or scene-graph reasoning [2, 6, 23]; and MomaGraph studies task-oriented structured intermediates with learned graph policies [17]. TaskGround shares the motivation of using structured intermediates, but differs by targeting the full-scene information condition and by explicitly inferring and completing instance-grounded executable task structure in a training-free, model-agnostic pipeline before skill-level execution.

### 6.2 Embodied Household Benchmarks and Evaluation Protocols

Embodied household benchmarks and simulators provide executable environments for evaluating household agents. VirtualHome represents household activities with programmatic action specifications [28], while BEHAVIOR and BEHAVIOR-1K define activities through predicate-based initial and goal conditions [19, 38]. Recent embodied-agent interfaces and planning benchmarks further evaluate LLM-based agents with structured inputs, action spaces, constraints, or evaluation functions [5, 7, 20]. Instruction-following and dialogue benchmarks such as ALFRED, ALFWorld, TEACh, and DialFRED test grounded execution and interactive assistance [11, 26, 34, 35]. Beyond these simulator-backed and instruction-following settings, recent benchmarks have moved toward more user-facing household scenarios, including personalized memory use [18], vague referring expressions in robot task planning [16], and smart-home evaluation with contextual or personalized instructions [21, 40]. These works study important factors beyond clean instruction following, with different emphases on memory, reference ambiguity, device control, or planner evaluation. In contrast, **FullHome** evaluates the full-scene information condition: agents receive a complete household scene and a situated household request, but not task-relevant object lists, explicit goals, or process constraints. It measures whether agents can infer executable task structure before execution.

## 7 Discussion

Our results show that full-scene household reasoning is not merely a long-context action-generation problem: agents must identify task-relevant scene context, infer implicit goals and process requirements, and produce executable skill-level actions. TaskGround separates these difficulties into scene grounding, task-structure inference with completion, and skill-level execution, yielding consistent gains across model families. Our analyses also show that high-recall grounding alone is insufficient, while oracle scene-slice results indicate room for improvement in grounding and downstream task reasoning. The fixed completion priors may require adaptation to household-specific routines. FullHome is a controlled simulator-backed setting with structured scenes and predefined skills; future work should extend this setting to perception noise, partial observability, clarification dialogue, household preferences, safety monitoring, and robust failure recovery.

## 8 Conclusion

We formalized **full-scene household reasoning**, where agents must act from a complete household scene and a situated household request rather than a curated task specification. To tackle this problem, we proposed **TaskGround**, a training-free *Ground–Infer–Execute* framework that grounds complete scenes, infers and completes executable task structure, and compiles it into grounded skill-level actions. We also introduced **FullHome**, a simulator-backed evaluation suite for systematically measuring this setting. Experiments show that TaskGround consistently improves task success across proprietary and open-weight models and enables compact models to approach frontier-model performance under direct complete-scene prompting with lower total input-token cost. These results identify executable task-structure inference as a central bottleneck in task-level household reasoning.

## References


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

[2] Vineet Bhat, Ali Umut Kaypak, Prashanth Krishnamurthy, Ramesh Karri, and Farshad Khorrami. Grounding llms for robot task planning using closed-loop state feedback, 2024. *URL https://arxiv. org/abs/2402.08546*, 2024.

[3] Meghan Booker, Grayson Byrd, Bethany Kemp, Aurora Schmidt, and Corban Rivera. Embodiedrag: Dynamic 3d scene graph retrieval for efficient and scalable robot task planning. *arXiv preprint arXiv:2410.23968*, 2024.

[4] Joseph Bugeja, Andreas Jacobsson, and Paul Davidsson. Prash: a framework for privacy risk analysis of smart homes. *Sensors*, 21(19):6399, 2021.

[5] Matthew Chang, Gunjan Chhablani, Alexander Clegg, Mikael Dallaire Cote, Ruta Desai, Michal Hlavac, Vladimir Karashchuk, Jacob Krantz, Roozbeh Mottaghi, Priyam Parashar, et al. Partnr: A benchmark for planning and reasoning in embodied multi-agent tasks. *arXiv preprint arXiv:2411.00081*, 2024.

[6] Yiye Chen, Harpreet S Sawhney, Nicholas Gydé, Yanan Jian, Jack Saunders, Patricio Vela, and Benjamin E Lundell. sg2. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 30332–30340, 2026.

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

[8] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

[9] DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

[10] Tamara Denning, Cynthia Matuszek, Karl Koscher, Joshua R Smith, and Tadayoshi Kohno. A spotlight on security and privacy risks with future household robots: attacks and lessons. In *Proceedings of the 11th international conference on Ubiquitous computing*, pages 105–114, 2009.

[11] Xiaofeng Gao, Qiaozi Gao, Ran Gong, Kaixiang Lin, Govind Thattai, and Gaurav S Sukhatme. Dialfred: Dialogue-enabled agents for embodied instruction following. *IEEE Robotics and Automation Letters*, 7(4):10049–10056, 2022.

[12] Jonas Herzog, Jiangpin Liu, and Yue Wang. Domain-conditioned scene graphs for state-grounded task planning. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4142–4149. IEEE, 2025.

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

[14] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

[15] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[16] Chenxi Jiang, Chuhao Zhou, and Jianfei Yang. Rei-bench: Can embodied agents understand vague human instructions in task planning? *arXiv preprint arXiv:2505.10872*, 2025.

[17] Yuanchen Ju, Yongyuan Liang, Yen-Jen Wang, Nandiraju Gireesh, Yuanliang Ju, Seungjae Lee, Qiao Gu, Elvis Hsieh, Furong Huang, and Koushil Sreenath. Momagraph: State-aware unified scene graphs with vision-language model for embodied task planning. *arXiv preprint arXiv:2512.16909*, 2025.

[18] Taeyoon Kwon, Dongwook Choi, Hyojun Kim, Sunghwan Kim, Seungjun Moon, Beongwoo Kwak, Kuan-Hao Huang, and Jinyoung Yeo. Embodied agents meet personalization: Investigating challenges and solutions through the lens of memory utilization. *arXiv preprint arXiv:2505.16348*, 2025.

[19] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, et al. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning*, pages 80–93. PMLR, 2023.

[20] Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Li E Li, Ruohan Zhang, et al. Embodied agent interface: Benchmarking llms for embodied decision making. *Advances in Neural Information Processing Systems*, 37: 100428–100534, 2024.

[21] Silin Li, Yuhang Guo, Jiashu Yao, Zeming Liu, and Haifeng Wang. Homebench: Evaluating llms in smart homes with valid and invalid instructions across single and multiple devices. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12230–12250, 2025.

[22] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International conference on robotics and automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[23] Anatoly O Onishchenko, Alexey K Kovalev, and Aleksandr I Panov. Lookplangraph: Embodied instruction following method with vlm graph augmentation. *arXiv preprint arXiv:2512.21243*, 2025.

[24] OpenAI. Introducing GPT-4.1 in the api. `https://openai.com/index/gpt-4-1/`, April 2025. Accessed: 2025-09-12.

[25] OpenAI. GPT-5 system card. `https://cdn.openai.com/gpt-5-system-card.pdf`, August 2025. Accessed: 2025-09-24.

[26] Aishwarya Padmakumar, Jesse Thomason, Ayush Shrivastava, Patrick Lange, Anjali Narayan-Chen, Spandana Gella, Robinson Piramuthu, Gokhan Tur, and Dilek Hakkani-Tur. Teach: Task-driven embodied agents that chat. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 2017–2025, 2022.

[27] Ugo Pagallo. Robots in the cloud with privacy: A new threat to data protection? *Computer Law & Security Review*, 29(5):501–508, 2013.

[28] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8494–8502, 2018.

[29] Guanqiao Qu, Qiyuan Chen, Wei Wei, Zheng Lin, Xianhao Chen, and Kaibin Huang. Mobile edge intelligence for large language models: A contemporary survey. *IEEE Communications Surveys & Tutorials*, 27(6):3820–3860, 2025.

[30] Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL `https://qwen.ai/blog?id=qwen3.5`.

[31] 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. *arXiv preprint arXiv:2307.06135*, 2023.

[32] Allen Z Ren, Anushri Dixit, Alexandra Bodrova, Sumeet Singh, Stephen Tu, Noah Brown, Peng Xu, Leila Takayama, Fei Xia, Jake Varley, et al. Robots that ask for help: Uncertainty alignment for large language model planners. *arXiv preprint arXiv:2307.01928*, 2023.

[33] Bokui Shen, Fei Xia, Chengshu Li, Roberto Martín-Martín, Linxi Fan, Guanzhi Wang, Claudia Pérez-D'Arpino, Shyamal Buch, Sanjana Srivastava, Lyne Tchapmi, et al. igibson 1.0: A simulation environment for interactive tasks in large realistic scenes. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7520–7527. IEEE, 2021.

[34] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749, 2020.

[35] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

[36] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022.

[37] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2998–3009, 2023.

[38] Sanjana Srivastava, Chengshu Li, Michael Lingelbach, Roberto Martín-Martín, Fei Xia, Kent Elliott Vainio, Zheng Lian, Cem Gokmen, Shyamal Buch, Karen Liu, et al. Behavior: Benchmark for everyday household activities in virtual, interactive, and ecological environments. In *Conference on robot learning*, pages 477–490. PMLR, 2022.

[39] Gemma Team. Gemma 3. 2025. URL `https://goo.gle/Gemma3Report`.

[40] Nikhil Verma, InJung Yang, Sungil Kim, KoKeun Kim, YoungJoon Kim, Manasa Bharadwaj, Yolanda Liu, and Kevin Ferreira. Personalhomebench: Evaluating agents in personalized smart homes. *arXiv preprint arXiv:2604.16813*, 2026.

[41] Rui Wang, Zhiyong Gao, Liuyang Zhang, Shuaibing Yue, and Ziyi Gao. Empowering large language models to edge intelligence: A survey of edge efficient llms and techniques. *Computer Science Review*, 57:100755, 2025.

[42] Shu Wang, Muzhi Han, Ziyuan Jiao, Zeyu Zhang, Ying Nian Wu, Song-Chun Zhu, and Hangxin Liu. Llm^ 3: Large language model-based task and motion planning with motion failure reasoning. In *2024 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 12086–12092. IEEE, 2024.

[43] Maximiliane Windl, Jan Leusmann, Albrecht Schmidt, Sebastian S Feger, and Sven Mayer. Privacy communication patterns for domestic robots. In *Twentieth Symposium on Usable Privacy and Security (SOUPS 2024)*, pages 121–138, 2024.

[44] Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas Funkhouser. Tidybot: Personalized robot assistance with large language models. *Autonomous Robots*, 47(8):1087–1102, 2023.

[45] Yike Wu, Jiatao Zhang, Nan Hu, Lanling Tang, Guilin Qi, Jun Shao, Jie Ren, and Wei Song. Mldt: Multi-level decomposition for complex long-horizon robotic task planning with open-source large language model. In *International Conference on Database Systems for Advanced Applications*, pages 251–267. Springer, 2024.

[46] Zhenyu Wu, Ziwei Wang, Xiuwei Xu, Jiwen Lu, and Haibin Yan. Embodied task planning with large language models. *arXiv preprint arXiv:2307.01848*, 2023.

[47] Bangjun Xiao, Bingquan Xia, Bo Yang, Bofei Gao, Bowen Shen, Chen Zhang, Chenhong He, Chiheng Lou, Fuli Luo, Gang Wang, et al. Mimo-v2-flash technical report. *arXiv preprint arXiv:2601.02780*, 2026.

[48] Haoming Ye, Yunxiao Xiao, Cewu Lu, and Panpan Cai. Uniplan: Vision-language task planning for mobile manipulation with unified pddl formulation. *arXiv preprint arXiv:2602.08537*, 2026.

[49] Yue Zheng, Yuhao Chen, Bin Qian, Xiufang Shi, Yuanchao Shu, and Jiming Chen. A review on edge large language models: Design, execution, and applications. *ACM Computing Surveys*, 57 (8):1–35, 2025.

## 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 summarize the proposed full-scene household reasoning setting, the TaskGround framework, the FullHome evaluation suite, and the main empirical findings. These claims are supported by the problem formulation, framework description, benchmark protocol, and experiments reported in the paper.

   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: The Discussion section describes limitations of the controlled simulator-backed setting, including structured scene representations, predefined skill-level actions, and the gap to real-world deployment with perception noise, partial observability, safety monitoring, and human intervention.

   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 does not present theoretical results, theorems, or formal proofs. The mathematical notation is used to define the task setting and framework components.

   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 and appendix describe the benchmark construction, task subsets, evaluation condition, model set, serving setup, decoding settings, output parsing, simulator execution protocol, and metric computation.

   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: [No]

   Justification: At submission time, we do not include the full benchmark files and evaluation code. To support verification and reproducibility, the paper and appendix provide the benchmark construction pipeline, task statistics, evaluation protocol, prompt and parsing details, simulator execution procedures, and model settings. We plan to release the FullHome task specifications and evaluation code after the review process.

   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: The experimental section specifies the FullHome task subsets, input condition, compared models, evaluation metrics, and main baselines. The appendix provides model serving, decoding settings, automatic parsing, simulator execution, and grounding diagnostics. No model training or fine-tuning is performed.

   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: [No]

Justification: We report success rates on fixed benchmark splits with decoding temperature set to zero, but do not report repeated-run error bars, confidence intervals, or formal significance tests.

- 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: [No]

   Justification: We report hardware configurations, serving setups, and input-token budgets, but do not provide complete per-run timing or total compute estimates.

   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 research uses simulator-backed household environments and symbolic task specifications. It does not involve deployment on real users, collection of personal household data, or human-subject experiments, and we believe 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: [Yes]

    Justification: The paper discusses potential positive impacts, such as more reliable and privacy-conscious household robot reasoning, and potential risks, including incorrect action execution, over-reliance on automated household agents, and privacy concerns associated with detailed household scene states.

    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 a high-risk pretrained model, image generator, scraped web dataset, or asset with substantial misuse potential. FullHome is based on simulator-derived household scenes, situated requests, and symbolic task specifications.

    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: We cite the original sources for the simulators, environments, models, and software tools used in the paper. The appendix lists the existing assets used and their roles in benchmark construction, evaluation, and model inference, together with license or terms-of-use information where available.

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 paper introduces FullHome, a new evaluation suite consisting of task definitions, situated household requests, hidden task structures, and simulator-backed evaluation protocols. The paper and appendix document its construction pipeline, evaluation condition, subset statistics, coverage, quality control, intended use, and limitations.

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 paper does not involve crowdsourcing experiments, external annotators, or human-subject studies. Human validation refers to author-side task curation and quality checking of simulator-based task specifications, without collecting personal data or studying participant behavior.

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: The paper does not involve human-subject experiments, collection of personal data, or interaction with study participants. Therefore, IRB approval or equivalent review is not applicable.

    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: LLMs are central to both the proposed framework and the experimental evaluation. The paper describes their use in scene grounding, task-structure inference, direct-action baselines, LLM-assisted task drafting during benchmark construction, and model comparisons, and the appendix reports evaluated model versions, serving setups, and decoding settings.

    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.

# A Appendix

## Table of Contents


### Benchmark and Method Details

**Appendix B: Benchmark Details** . . . . . **23**

Describes the construction and validation of FULLHOME, including the evaluation condition, task subsets, construction yield, simulator-in-the-loop construction pipeline, and coverage statistics.

**Appendix C: TaskGround Details** . . . . . **26**

Provides implementation details of TASKGROUND, including the Ground–Infer–Execute decomposition, scene grounding, task-structure inference and completion, skill-level execution, and total input-token cost.

**Appendix D: Prompt Templates** . . . . . **28**

Lists the prompt templates used for scene grounding, task-structure inference, ordering-sensitive goal prediction, and direct action sequencing baselines.

### Experiments and Analysis

**Appendix E: Experiment Details** . . . . . **31**

Provides additional details on evaluated models, serving infrastructure, output parsing and failure handling, VirtualHome and BEHAVIOR execution, and scene grounding diagnostics.

**Appendix F: Qualitative Examples and Failure Cases** . . . . . **33**

Presents representative qualitative examples and recurrent failure modes of direct full-scene action generation, together with how TASKGROUND mitigates them through grounding, task-structure inference, completion, and skill-level execution.

### Reproducibility and Responsible Use

**Appendix G: Broader Impacts** . . . . . **36**

Discusses potential benefits, risks, and mitigation considerations for full-scene household reasoning and household embodied agents.

**Appendix H: Compute Resources** . . . . . **37**

Reports compute resources used for local open-weight model inference, provider-served model evaluation, simulator execution, and automatic evaluation.

**Appendix I: Asset Licenses** . . . . . **38**

Summarizes external assets used in this work, including household simulators, benchmark environments, model providers, open-weight models, and software dependencies.

Table 3: Construction yield of FULLHOME. We over-generate candidate tasks and retain only those that pass simulator validation, process differentiation when applicable, and human review.

| Subset | Candidates | Released | Yield |
|---|---|---|---|
| VG-200 | 300 | 200 | 66.7% |
| VP-100 | 300 | 100 | 33.3% |
| BG-65 | 120 | 65 | 54.2% |
| BP-35 | 120 | 35 | 29.2% |
| Total | 840 | 400 | 47.6% |

# B Benchmark Details

This section provides additional details on the construction and validation of FULLHOME, the evaluation suite used to study full-scene household reasoning. Each task instance provides a *complete household scene* and a *situated household request*, and the agent must output a *grounded skill-level action sequence*. The hidden task structure, including final-state goals and process constraints when applicable, is used only for evaluation and is not provided to the agent.

FULLHOME contains 400 carefully designed and human-validated household tasks built on executable household simulation environments, including VirtualHome [28] and BEHAVIOR [38]. The benchmark is constructed through an over-generation and quality-gating pipeline: candidate tasks are generated from simulator-grounded scene inventories and household scenarios, and only tasks that pass simulator validation and human review are retained.

## B.1 Evaluation Condition

FULLHOME changes the test-time information condition of existing simulator-backed household evaluations. The agent receives only the initial complete household scene $\mathcal{S}_0$ and the situated household request $u$, and must output a grounded skill-level action sequence $a_{1:T}$. It is not given task-relevant object lists, candidate action arguments, explicit goals, target states, process constraints, or reference action sequences. Thus, the agent must infer task-relevant scene elements, intended goal conditions, and process requirements before producing executable grounded actions.

The benchmark contains four subsets:

- **VG-200**: 200 VirtualHome goal tasks, evaluated by final-state satisfaction.
- **VP-100**: 100 VirtualHome process tasks, evaluated by both final-state and process-constraint satisfaction.
- **BG-65**: 65 BEHAVIOR goal tasks, used to evaluate cross-environment generalization.
- **BP-35**: 35 BEHAVIOR process tasks, evaluated by both executable goals and process constraints.

Here, **V/B** denote the VirtualHome and BEHAVIOR subsets, and **G/P** denote goal and process tasks.

## B.2 Construction Pipeline

We construct FULLHOME through a simulator-in-the-loop pipeline that over-generates candidate tasks and then filters them through automatic and human validation. The goal is to obtain situated household tasks that are executable, unambiguous, and automatically evaluable, while still requiring agents to infer what should be done from a complete household scene and a situated request.

**Stage 1: Scene inventory extraction.** We first extract the scene inventory from each simulator environment. For VirtualHome, we use the evolving-graph representation to obtain rooms, objects, object states, affordances, and relations. For BEHAVIOR, we use object-rich multi-room household scenes and parse the corresponding scene assets and task specifications to obtain rooms, fixed furniture, sampleable objects, supported predicates, and object affordances. This produces a structured inventory for each scene, which serves as the basis for task design and is also the complete household scene provided to evaluated agents.

Table 4: Scene, room, and theme coverage of FULLHOME. For VirtualHome, scenes correspond to functional rooms within the apartment graph; for BEHAVIOR, scenes correspond to (BDDL activity, house) base configurations. Themes are the per-task LLM-generated descriptions retained after the construction pipeline; each task corresponds to one theme on BEHAVIOR, while VirtualHome uses a smaller theme pool with multiple tasks per theme.

| Subset | #Tasks | #Houses | #Scenes | #Themes |
|---|---|---|---|---|
| VG-200 | 200 | 1 | 4 rooms | 98 |
| VP-100 | 100 | 1 | 4 rooms | 87 |
| BG-65 | 65 | 4 | 10 base scenes | 65 |
| BP-35 | 35 | 3 | 9 base scenes | 34 |
| Union | 400 | **5** | **20** | ∼280 |

**Stage 2: Household scenario design.** We design household scenarios to cover common daily activities and deployment contexts, such as cleaning, meal preparation, object organization, workspace setup, guest preparation, appliance use, lighting, rest, and safety routines. The scenario design is grounded in the extracted scene inventory: candidate tasks are required to refer only to entities, states, and affordances available in the corresponding household scene. This prevents tasks from depending on hallucinated objects or unsupported actions.

**Stage 3: Situated request and hidden task-structure authoring.** For each scenario, we author a situated household request together with its hidden task structure. The request is written from the user's perspective and is designed to be natural rather than a clean task specification. It may leave task-relevant objects, target states, or ordering requirements implicit, but the intended task must be recoverable from the complete scene context. We annotate the corresponding hidden task structure $\tau = (\mathcal{G}, \mathcal{P})$, where $\mathcal{G}$ contains final-state goal predicates and $\mathcal{P}$ contains process constraints when applicable. Both goals and process constraints are grounded to entities in the simulator scene and are never included in the model input.

**Stage 4: Ground-truth execution and simulator validation.** Each candidate task is paired with a ground-truth skill-level execution. We generate candidate executions using rule-based templates for canonical household motifs and LLM-assisted drafting for long-tail scenarios. The resulting action sequence is executed and checked in the corresponding simulator. A task is retained only if the execution succeeds and the final state satisfies all annotated goal predicates. For process tasks, the execution trajectory must also satisfy the annotated process constraints.

For VirtualHome, we validate executions with its deterministic evolving-graph transition system. For BEHAVIOR, whose underlying simulator can involve continuous physics and online object instantiation, we instantiate the candidate scene and snapshot the resulting object configuration, including object states, parent-child relations, and predicate truth values. We then run a deterministic graph-style satisfaction check over the snapshot. This preserves BEHAVIOR task semantics while making evaluation reproducible.

**Stage 5: Process differentiation.** For process tasks, we additionally verify that process constraints are non-trivial. We check that the annotated process requirements distinguish valid executions from alternative executions that may reach the same final state but violate the required ordering or miss an intermediate condition. This ensures that process-task success is not reducible to final-state goal satisfaction alone.

**Stage 6: Human review and final filtering.** All surviving tasks are reviewed for request naturalness, clarity, grounding consistency, executability, and alignment between the situated request and the hidden task structure. We remove tasks with ambiguous intent, unsupported object references, unnatural phrasing, invalid simulator behavior, or mismatches between the request and the annotated goals or process constraints. The final benchmark is sampled after validation to maintain diversity across environments, rooms, household scenarios, and task types.

Table 3 summarizes the construction funnel. Starting from 840 candidate tasks, the pipeline retains 400 tasks after simulator validation, process differentiation when applicable, and human review. The lower yield of process tasks reflects the additional ordering and intermediate-state requirements imposed by $\mathcal{P}$.

### B.3 Coverage Statistics

Table 4 summarizes the scene, room, and theme coverage of FULLHOME. The VirtualHome subsets use a single underlying household environment, but each task is constructed on a per-task scene perturbation that varies object placements, states, and clutter, supporting controlled analysis in a known-home setting. The BEHAVIOR subsets span four iGibson houses and ten distinct (BDDL activity, house) base scenes, providing a complementary cross-environment evaluation setting with realistic object layouts and physics-grounded affordances.

The task themes span a wide range of household contexts, including hygiene, meal preparation, cleanup, workspace setup, guest preparation, appliance use, lighting and ambience, rest, and safety routines. Process tasks additionally annotate ordering invariants drawn from common household routines, such as cleaning before serving, preparing a surface before placing objects on it, loading before running an appliance, and closing a container or appliance before activation.

# C TaskGround Details

This section provides implementation details of TASKGROUND that are omitted from the main text. Given a complete household scene $\mathcal{S}_0$ and a situated household request $u$, TASKGROUND follows the same three-stage *Ground–Infer–Execute* decomposition defined in Sec. 3:

$$\text{ground: } \mathcal{S}_r = R(\mathcal{S}_0, u), \quad \text{infer: } \hat{\mathbf{g}} = T(\mathcal{S}_r, u),\ \tilde{\tau} = C(\hat{\mathbf{g}}), \quad \text{execute: } a_{1:T} = E(\tilde{\tau}, \mathcal{S}_r).$$

Here, $R$ is the scene grounder, $T$ is the task-structure inference module, $C$ is the completion module, and $E$ is the executor. The output of $T$, $\hat{\mathbf{g}}$, is an initially inferred ordered goal sequence. The output of $C$, $\tilde{\tau}$, is the completed executable task structure used by the executor. The scene grounder $R$ and task-structure inference module $T$ each make one LLM call, while $C$ and $E$ are deterministic. At test time, TASKGROUND never accesses hidden final-state goals, hidden process constraints, reference action sequences, task identifiers, or evaluation outcomes. The same prompts are used across models, with decoding temperature set to 0.

## C.1 Ground: Scene Grounder

The grounder produces a compact task-relevant scene slice $\mathcal{S}_r \subseteq \mathcal{S}_0$. It is designed to reduce the input cost of downstream reasoning while preserving the entities, states, affordances, and local relations needed for executable task-structure inference.

**Entity inventory.** The complete scene is first converted into a lightweight entity inventory. For each room, we list object classes and relevant state summaries, such as whether an object is open, closed, dirty, clean, on, off, or receptacle-like. This inventory is substantially shorter than the full scene graph because it does not serialize all pairwise object relations.

**Entity selection.** The LLM receives the situated request $u$ and the entity inventory extracted from $\mathcal{S}_0$. It selects potentially task-relevant room–object-class pairs. The prompt is recall-oriented: the model is instructed to include explicitly mentioned entities, implicit tools, destinations, containers, surfaces, and objects that may be required by the household context.

**Scene-slice reconstruction.** Selected room–object-class pairs are resolved to all matching scene-graph nodes $\hat{\mathcal{V}}_r$ by deterministic lookup. Unmatched selections are discarded, so the reconstructed slice contains only existing scene-graph nodes. We then expand the selected node set with structural context needed for execution, including containing rooms, the agent node and state, relevant containers or surfaces, object states, affordances, and local relations. If $\mathcal{V}_r$ is the expanded node set, the grounded slice is the induced subgraph

$$\mathcal{S}_r = (\mathcal{V}_r, \mathcal{E}_r), \qquad \mathcal{E}_r = \{(v_i, v_j) \in \mathcal{E}_0 \mid v_i, v_j \in \mathcal{V}_r\}.$$

Thus, the LLM selects task-relevant room–object-class pairs, while nodes and relations are recovered deterministically from the original scene graph.

## C.2 Infer: Task-Structure Inference and Completion

Given the grounded scene slice $\mathcal{S}_r$ and request $u$, the inference module $T$ predicts an ordered goal sequence

$$\hat{\mathbf{g}} = (\hat{g}_1, \ldots, \hat{g}_M).$$

Each goal atom describes a desired state or relation over grounded entities, such as `CLEAN(table)`, `ON(cup, table)`, `CLOSED(dishwasher)`, or `INSIDE(cup, dishwasher)`. This goal-level interface asks the LLM to infer what should be achieved, rather than to directly generate a long skill-level action sequence.

For VirtualHome, goal atoms are represented using node-state and edge-relation predicates:

```
{
  "goals": [
    {"predicate": "CLEAN",  "args": [<id>]},
    {"predicate": "ON",     "args": [<from_id>, <to_id>]},
    {"predicate": "INSIDE", "args": [<from_id>, <to_id>]}
  ]
}
```

Allowed state predicates include `OPEN`, `CLOSED`, `CLEAN`, `ON`, `OFF`, `PLUGGED_IN`, and `PLUGGED_OUT`, and allowed relation predicates include `ON` and `INSIDE`. For BEHAVIOR, the same goal-level interface is mapped to BDDL-supported predicates. The model is required to use only entities present in $\mathcal{S}_r$. Malformed JSON is recovered when possible, but unsupported predicates, duplicate goals, initially satisfied goals, and references to non-existent entity identifiers are rejected automatically.

The completion module $C$ then transforms the initial goal sequence $\hat{\mathbf{g}}$ into a completed executable task structure

$$\tilde{\tau} = (\tilde{g}_1, \ldots, \tilde{g}_K).$$

The completion rules encode fixed household priors over goal patterns, such as closing an appliance before turning it on, closing a container after insertion, or cleaning a dirty surface before placing an object on it. These rules are specified before evaluation and do not use hidden goals, hidden process constraints, reference actions, task identifiers, or evaluation outcomes. Thus, $C$ is a deterministic test-time completion step rather than an oracle correction. Its output $\tilde{\tau}$ is an ordered goal sequence that contains both final-state goals and process-critical intermediate goals.

### C.3 Execute: Skill-Level Executor

The executor $E$ compiles the completed executable task structure $\tilde{\tau}$ into a grounded skill-level action sequence $a_{1:T}$. It treats each ordered goal atom as a desired scene-state change and realizes it using the available household skills, such as navigation, manipulation, cleaning, container operation, and appliance control. For example, a goal such as `CLEAN(table)` may be realized as a sequence that navigates to a cleaning tool, grasps it, walks to the table, and wipes the table.

The executor maintains an internal symbolic state after each emitted action so that later goals are planned against the latest scene state. A precondition layer inserts generic steps required for executable actions, such as navigating near an object before manipulation, holding an object before placing it, opening a closed container before insertion or retrieval, and ensuring that the agent has a free hand before grasping. When $\tilde{\tau}$ contains ordered intermediate goals, the executor preserves this ordering. It does not infer new task goals or process requirements; it only realizes the completed structure produced by the previous stage.

### C.4 Token and Cost Budget

We measure input-token cost using the `gpt-4o` BPE tokenizer over the actual prompts used in our experiments. The naive baseline serializes the complete household scene in a single LLM call. TASKGROUND uses two LLM calls per task: a grounding call over a lightweight entity inventory and an inference call over the grounded scene slice. Table 5 reports the total input-token cost summed over all LLM calls per task. The same serialized prompts are reused across evaluated models, so the reported costs reflect the scale of the scene representation rather than model-specific prompt changes.

Table 5: Total input-token cost of TASKGROUND versus the naive full-scene baseline on FULLHOME, summed over all LLM calls per task. Tokens are measured with the `gpt-4o` BPE tokenizer over the actual prompts.

| Env. | Subset | Naive | TASKGROUND | | Reduction |
|---|---|---|---|---|---|
| | | 1 call | Ground | Infer | |
| VirtualHome | VG-200 | ∼84,300 | ∼2,030 | ∼2,680 | **17.9**× |
| | VP-100 | ∼84,300 | ∼2,030 | ∼2,660 | **18.0**× |
| BEHAVIOR | BG-65 | ∼30,800 | ∼2,000 | ∼3,300 | **5.8**× |
| | BP-35 | ∼23,800 | ∼2,000 | ∼4,100 | **3.9**× |

The reduction varies with the scale of the underlying household scene, but TASKGROUND consistently lowers total input-token cost by replacing complete-scene serialization with a lightweight grounding call followed by task-structure inference over a compact scene slice.

# D Prompt Templates

This appendix provides the prompt templates used in our evaluation to support reproducibility. Placeholders are shown in {braces}. For all prompting-based stages, models are instructed to use only the entities, states, relations, and action primitives provided in the input. The prompts do not include hidden final-state goals, process constraints, oracle actions, task identifiers, or evaluation outcomes.

## D.1 Scene Grounding Prompt

The scene grounding prompt receives the situated household request and a room-grouped object catalog derived from the complete household scene. It asks the model to identify task-relevant object classes and rooms. The system then resolves selected class names to scene-graph node IDs and applies structural expansion, such as adding relevant rooms, containers, surfaces, power-related objects, and local relational context.

**Scene Grounding Prompt**

```
You are a scene retriever for an embodied household agent.

REQUEST:
{request}

SCENE OBJECTS, grouped by room. Each line contains: class_name [count] |
states | properties.
{catalog}

Task: decompose the request into individual goals or requirements. For
each goal, list the (room, object_class) pairs the robot needs to read or
interact with. You do not need to enumerate object IDs or count instances.
The system will automatically include all matching objects in that room
and add containers, surfaces, rooms, and local context through structural
expansion.

GUIDELINES:
- object_class must be the exact class_name shown in the scene table.
- room must be the exact room name. Use "any" only if the object's room
cannot be determined from the request.
- Include implicit tools even if they are not mentioned by name:
wipe or clean a surface -> include rag or towel.
wash an object -> include sink and faucet.
- Include implicit objects for high-level intents:
set up for work -> include computer, keyboard, and mouse if present.
- Include destinations for placement requests. For "place A on/in B", list B
as well.
- Be inclusive: missing a relevant object may cause task failure.

Output only a JSON object with the following format:
{"goals": [{"goal": "...", "selections": [{"room": "...", "objects":
[...]}]}]}
```

## D.2 Task-Structure Inference Prompt

The task-structure inference prompt receives the situated household request and the grounded scene slice produced by the scene grounding stage. It asks the model to infer the executable task structure, including final-state goals and process-relevant ordering requirements, using only entities in the grounded scene slice.

**Task-Structure Inference Prompt**

```
You are a household robot planner. Given a situated household request
and the current grounded scene graph, infer the target goal states and
process-relevant requirements needed to complete the task.
```

```
Output two types of final-state goals:
1. Node goals: an object should reach a specific state.
2. Edge goals: an object should be moved to a specific location or
container.

Possible target states for node goals:
ON, OFF, OPEN, CLOSED, CLEAN, PLUGGED_IN, PLUGGED_OUT

Possible relations for edge goals:
ON, INSIDE

Rules:
1. Only include changes from the current state.
2. Use exact object IDs from the grounded scene graph.
3. Include all goals implied by the request and the household context.
4. Do not invent objects that are not present in the grounded scene graph.
5. If ordering is important, list goals in the order they should be
achieved.

REQUEST:
{request}

NODES:
{nodes}

EDGES:
{edges}

Output only a JSON object with the following format:
{"node_goals": [...], "edge_goals": [...]}
```

### D.3 Ordering Hint for Goal Sequences

For the goal-sequence variant used by TaskGround, the following ordering hint is appended to the task-structure inference prompt.

**Ordering Hint**

```
IMPORTANT: ORDER MATTERS.
List node_goals and edge_goals in the order they should be achieved. Each
goal will be executed sequentially against the current scene state. For
tasks where order is meaningful, such as cleaning before placing, opening
a container before putting an object inside, or preparing an object before
serving it, put earlier prerequisites first. For order-independent tasks,
any consistent order is acceptable.
```

### D.4 Action Sequencing Prompt for Direct-Action Baselines

The action sequencing prompt is used by direct-action baselines, including Naive and Ground+Act. In these settings, the model directly outputs a grounded skill-level action sequence instead of first inferring executable task structure.

**Action Sequencing Prompt**

```
The task is to guide the robot to take actions from the current scene state
based on a situated household request.

Unlike goal-conditioned settings, the input does not explicitly provide
node goals, edge goals, process constraints, or oracle actions. You must
infer the intended target objects, target states, and target relations from
the request and the provided scene graph, and then output a valid grounded
skill-level action sequence.

Supported actions include:
CLOSE, GRAB, OPEN, PUTBACK, PUTIN, SWITCHON, SWITCHOFF, WIPE, WASH, PLUGIN,
```

```
PLUGOUT, WALK.

Key rules:
- WALK to an object before interacting with it.
- GRAB requires a free hand.  If both hands are full, release an object first.
- WIPE requires holding a cleaning tool when applicable.
- SWITCHON requires the object to be OFF and PLUGGED_IN when applicable.
- Use only object IDs that appear in the provided scene graph.

REQUEST:
{request}

NODES:
{nodes}

EDGES:
{edges}

Output only a JSON list of action commands.
```

# E Experiment Details

This section provides additional experimental details for the results reported in the main paper. Across all experiments, each task provides a complete household scene and a situated household request. The agent must output a grounded skill-level action sequence; hidden final-state goals, process constraints, reference action sequences, task identifiers, and evaluation outcomes are never provided to the model. All prompting-based stages use `temperature=0`, and the same prompt templates are used across models.

## E.1 Evaluated Models and Serving Infrastructure

We evaluate both proprietary and open-weight models to cover a range of model capabilities and deployment assumptions. The proprietary group includes GPT-4o, GPT-4.1, Gemini-2.5-Flash, and GPT-5. The open-weight group includes DeepSeek-V4-Flash, MiMo-V2-Flash, Gemma-3-12B, and Qwen3.5-9B. This selection covers frontier proprietary systems as well as compact models that better reflect local-deployment constraints.

Table 6 summarizes the serving setup. Provider-served models are accessed through the corresponding model provider or routing service available at evaluation time. Locally served open-weight models are deployed with vLLM in bfloat16 precision; no model weights are updated during evaluation. For GPT-5, we use medium reasoning effort.

Table 6: Model zoo and serving infrastructure used in our experiments.

| Model | Access / Backend | Serving setup |
|---|---|---|
| GPT-4o | Azure OpenAI | Provider-served inference |
| GPT-4.1 | Azure OpenAI | Provider-served inference |
| GPT-5 | Azure OpenAI | Provider-served inference, medium reasoning effort |
| Gemini-2.5-Flash | Provider / routing service | Provider-served inference |
| DeepSeek-V4-Flash | Provider / routing service | Hosted open-weight inference |
| MiMo-V2-Flash | Provider / routing service | Hosted open-weight inference |
| Gemma-3-12B | vLLM local serving | 8× A100-40G or 4× A100-80G |
| Qwen3.5-9B | vLLM local serving | 8× A100-40G or 4× A100-80G |

## E.2 Output Parsing and Failure Handling

All model outputs are parsed automatically. For direct-action methods, the parser expects a grounded skill-level action sequence with supported action names and grounded entity arguments. For task-structure inference, the parser expects structured goal atoms over entities in the grounded scene slice. Manual correction is not applied during evaluation.

If an output contains malformed JSON, the parser attempts to extract the largest valid structured object when possible. If parsing still fails, the task is recorded as a parse failure. Outputs that reference nonexistent entities, unsupported predicates, invalid action names, or unsupported action-object combinations are counted as failures unless the issue can be deterministically normalized without changing task semantics. Transient provider or transport failures are retried up to three times; persistent failures are included in the denominator of all reported metrics.

## E.3 VirtualHome Execution

For the VirtualHome subsets, we use symbolic execution over evolving household scene graphs. The simulator updates the scene graph step by step as the predicted grounded skill-level action sequence is executed. An action may fail if it violates simulator preconditions, such as interacting with an object before walking near it, inserting an object into a closed container, or applying a switch action to a non-switchable object.

For goal tasks, success requires that the final scene state satisfies all annotated goal predicates $\mathcal{G}$. For process tasks, success additionally requires that the execution trajectory satisfies the annotated process constraints $\mathcal{P}$. All reported scores are computed over the full corresponding split.

### E.4 BEHAVIOR Execution

For the BEHAVIOR subsets, tasks are specified using simulator-supported household objects and executable predicate semantics. To make evaluation comparable across simulator backends, we convert the task state into a discrete graph abstraction whose nodes correspond to household entities and whose predicates encode states and relations such as containment, support, proximity, and object states.

We export and cache task snapshots using servers equipped with 4 NVIDIA RTX A6000 GPUs. Each snapshot records the scene objects, physical properties, room membership, and predicate-level state needed for prompt construction and evaluation. Predicted grounded skill-level action sequences are executed against this symbolic task state. Goal tasks are evaluated by final-state predicate satisfaction, while process tasks additionally check trajectory-level process constraints. This gives a unified evaluation interface across VirtualHome and BEHAVIOR while preserving each environment's object vocabulary and predicate semantics.

### E.5 Scene Grounding Diagnostics

For grounding diagnostics, we compare the grounded scene slice against oracle task-relevant nodes derived from the hidden task structure. Let $\mathcal{V}_r$ denote the node set retained in the grounded slice and $\mathcal{V}^\star$ denote the oracle task-relevant node set. We compute node recall as

$$\mathrm{NodeRecall} = \frac{|\mathcal{V}_r \cap \mathcal{V}^\star|}{|\mathcal{V}^\star|}.$$

This diagnostic is used only for analysis; final task performance is always evaluated by simulator execution.

# F Qualitative Examples and Failure Cases

We present qualitative examples from the FULLHOME evaluation set to illustrate typical failure modes of the naive *Full Scene + Act* baseline and how TASKGROUND mitigates them in selected cases. The examples highlight three recurring issues under complete household scenes: (i) *symbol hallucination* under large scene context, (ii) *affordance violations* where the model applies an operator to an object lacking the required property, and (iii) *implicit goal omission* where the model misses goals implied by the situated household request. These examples complement the aggregate results in Section 5 by showing how direct action generation can fail before or during simulator execution. In the selected examples, TASKGROUND mitigates these failures by grounding the complete household scene into a compact task-relevant scene slice, inferring executable task structure, completing missing goals or process-critical subgoals, and delegating action realization to a rule-based skill executor.

## F.1 Case 1: Symbol Hallucination Under Large Scene Context

**Task VG066: Situated request**

*"Hey, I'm settling in on the office couch for a long, sunny afternoon of reading and emails. Could you bring the bread from the dining area over to me and set it right on the couch so I don't have to get up? The little table by the couch feels grimy, and I want it quiet in there – no TV, just a comfy spot."*

**Ground-truth goals.** `ON(food_bread.3000, couch.352)`, `CLEAN(table.355)`, and `OFF(television.410)`.

**Failure modes under Full Scene + Act.**

- **GPT-4o**: SIM_FAIL. The model emits placeholder strings instead of valid integer IDs:

```
1. [WALK]    dining_area    'dining_area_id'    <-- hallucinated ID
2. [FIND]    bread          'bread_id'
3. [GRAB]    bread          'bread_id'
4. [WALK]    office_couch   'office_couch_id'
...
sim_error: invalid literal for int(): 'dining_area_id'
```

  The complete-scene context contains hundreds of valid IDs, but the model emits placeholder symbols that the simulator cannot dispatch.

- **GPT-4.1**: SIM_FAIL. The model violates the proximity precondition:

```
1. [WALK]      dining_room    201
2. [GRAB]      food_bread     3000          <-- character not close to bread
3. [WALK]      home_office    319
4. [PUTBACK]   food_bread     3000  couch  352
5. [SWITCHOFF] television     410
sim_error: <character>(65) is not close to <food_bread>(3000)
```

  The bread is inside the freezer, but the model attempts to grab it after walking only to the room-level node and omits the required `OPEN`.

- **GPT-5**: PASS. The model correctly opens the freezer first and produces a valid action sequence.

**Result.** The naive Full Scene + Act baseline fails for GPT-4o and GPT-4.1, while GPT-5 succeeds. Under TASKGROUND, all three models pass: the scene grounder reduces ID clutter, and the executor inserts generic precondition actions such as opening the freezer before grabbing the bread.

## F.2 Case 2: Affordance Violation on a Non-Switchable Object

**Task VG016: Situated request**

*"I'm calling it a night and heading to bed. Do a quick sweep so the place feels calm and dark with no screens left glowing, and the bedroom set for sleep. I'm not planning to read or work anymore."*

**Ground-truth goals.** `OFF(tablelamp.97)`, `OFF(tablelamp.98)`, `OFF(light.169)`, `OFF(light.245)`, `OFF(television.410)`, and `OFF(computer.417)`.

**Failure mode under Full Scene + Act.** Both GPT-4o and GPT-4.1 attempt to switch off `ceilinglamp.96`, which lacks the required `HAS_SWITCH` property:

```
1. [WALK]      ceilinglamp  96
2. [SWITCHOFF] ceilinglamp  96       <-- no HAS_SWITCH affordance
sim_error: <ceilinglamp>(96) does not have a switch
```

Although the full scene graph encodes object affordances, the direct-action baseline fails to filter candidates by the required property when many visually or semantically similar objects coexist. GPT-5 avoids the non-switchable lamp and passes.

**Result.** Under TASKGROUND, the scene grounder retrieves task-relevant switchable objects (table lamps, lights, television, computer) and the executor enforces action preconditions, so `SWITCHOFF` is never applied to objects without `HAS_SWITCH`. All selected models pass under TASKGROUND on this task.

### F.3 Case 3: Implicit Goal Omission (Schema-Level)

**Task VG133: Situated request**

*"Sun's out and I've only got a few minutes for lunch before my next call. I'm going to throw together a quick sandwich at the kitchen counter and sip some water at the dining table. Can you get that area looking ready so I can dig in right away?"*

**Ground-truth goals.** `CLEAN(table.226)`, `CLEAN(kitchen_counter.230)`, and `CLEAN(cup.2009)`.

**Failure modes under Full Scene + Act.**

- **GPT-4o**: GOAL_FAIL. The model wipes the surfaces but never washes the cup:

```
1. [WALK]  rag 3001
2. [GRAB]  rag 3001
3. [WALK]  kitchen_counter 230
4. [WIPE]  kitchen_counter 230
5. ...     wipes sink and table
           <-- never washes cup.2009
```

- **GPT-4.1**: GOAL_FAIL. The model produces a short action sequence that cleans surfaces but skips the cup.
- **GPT-5**: PASS. The model retrieves `cup.2009`, washes it at the faucet, and wipes both surfaces.

The phrase *"sip some water"* implies that a clean cup is needed, even though the cup is not explicitly named. The direct-action baseline captures the explicit surface-cleaning goals while missing this implicit, schema-level requirement.

**Result.** TASKGROUND mitigates this failure through task-structure inference and completion: the inference stage predicts explicit goals over the grounded scene slice, and the completion module augments goals implied by household schemas (e.g. "serve water" $\Rightarrow$ "clean cup"), enabling the executor to wash the cup before completion.

### F.4 Summary of Failure Modes

Table 7 summarizes recurrent failure modes of the naive Full Scene + Act baseline and how the stages of TASKGROUND mitigate them in the selected examples.

Table 7: Recurrent failure modes of the naive Full Scene + Act baseline and how TASKGROUND mitigates them in the selected examples.

| **Failure mode** | **Trigger** | **Mitigated by** |
|---|---|---|
| Symbol hallucination (SIM_FAIL) | The model emits placeholder strings instead of valid IDs available in the complete scene. | **Ground**: the compact scene slice reduces ID clutter. |
| Affordance violation (SIM_FAIL) | The model selects a similarly named object without the required property (e.g. `HAS_SWITCH`, `CAN_OPEN`). | **Ground + Execute**: grounding restricts candidates; the executor enforces affordance preconditions. |
| Missing precondition (SIM_FAIL) | The model forgets generic prerequisite steps (e.g. `OPEN` before `GRAB` from a container). | **Execute**: the rule-based executor inserts the missing precondition actions. |
| Implicit goal omission (GOAL_FAIL) | The model extracts explicit goals but misses goals implied by household schemas attached to the request. | **Infer + Completion**: schema-level completion adds missing goals and process-critical subgoals. |

## G Broader Impacts

This work studies full-scene household reasoning for embodied agents, with the goal of evaluating and improving an agent's ability to infer executable task structure from complete household scenes and situated household requests. A potential positive impact is that such evaluation can help develop household agents that better understand context-dependent user needs, avoid brittle execution from underspecified instructions, and operate more reliably in everyday indoor environments. By emphasizing compact open-weight models and structured grounding, the work also supports privacy-conscious and resource-efficient deployment settings, where detailed household states need not always be sent to frontier cloud-hosted proprietary models.

At the same time, household embodied agents raise important societal and safety concerns. Complete household scenes may contain sensitive information about personal spaces, routines, object usage, accessibility needs, or other private aspects of daily life. If such information is collected, stored, or transmitted without appropriate safeguards, it could create privacy risks. In addition, incorrect task-structure inference may lead to wrong or unsafe actions, especially when user requests are ambiguous, when the scene representation is incomplete or noisy, or when the robot acts in environments involving fragile objects, appliances, pets, children, or older adults. There is also a risk that household reasoning systems could be misused for intrusive monitoring or surveillance if deployed without proper consent and access control.

Our work does not deploy robots in real homes and does not claim to solve these issues. FullHome is intended as a controlled benchmark for studying task-structure inference under simulator-backed evaluation. Future deployment of similar systems should include privacy-preserving local inference where possible, clear user consent, secure handling of household scene data, uncertainty-aware planning, clarification dialogue for ambiguous requests, safety constraints, and human override mechanisms. We hope that explicitly evaluating full-scene reasoning and process constraints can help identify failures before such systems are considered for real-world household use.

## H Compute Resources

All experiments in this work are conducted in an inference-only setting. TaskGround is a training-free framework and does not require model fine-tuning or gradient-based optimization. No model weights are updated during our experiments, and no additional task-specific training data is used. Therefore, the reported compute resources correspond only to model inference, simulator execution, and automatic evaluation.

For locally served open-weight models, inference uses bfloat16 precision. In particular, Qwen3.5-9B is evaluated on GPU servers equipped with either 8 NVIDIA A100 GPUs with 40GB memory each or 4 NVIDIA A100 GPUs with 80GB memory each, depending on hardware availability. These resources support LLM calls for scene grounding and task-structure inference, as well as direct action generation in the baselines.

Simulator execution and automatic evaluation are performed separately from language-model inference. For BEHAVIOR, we instantiate iGibson-based scenes [33] and export task snapshots using servers equipped with 4 NVIDIA RTX A6000 GPUs. Predicted grounded skill-level action sequences are then evaluated symbolically over the exported scene states to verify final-state goals and process constraints.

For proprietary and hosted open-weight models, including GPT-family, Gemini-family, DeepSeek-family, and MiMo-family models, we use API-based inference through the corresponding model providers or API routing services available at evaluation time. These models are not run on local GPUs, and their compute resources are managed by the external API providers. For all models, we use the same benchmark inputs and evaluation protocol: each agent receives the initial complete household scene and the situated household request, and outputs a grounded skill-level action sequence for simulator-based execution and evaluation.

# I Asset Licenses

We use existing simulators, benchmarks, model APIs, and open-weight models only for research evaluation. We credit the original creators of these assets through citations in the main paper and appendix, and follow the corresponding licenses, model licenses, and terms of use. FullHome does not redistribute the original simulator assets or proprietary model weights. When our benchmark metadata or code is released, we will include copyright notices, references to the original assets, and license information for all derived files where applicable.

**Household simulation environments.** FullHome builds on existing household simulation environments and executable task interfaces. VirtualHome is used as a household activity simulation environment and is credited to its original authors [28]. The public VirtualHome repository is released under a permissive open-source license, and its Unity-related assets are accompanied by the license terms provided by the original project. BEHAVIOR is used as a household activity benchmark and executable task environment, and is credited to its original authors [38]. The public BEHAVIOR code repository is released under the MIT License, and we follow the license and usage conditions of the benchmark assets. For interactive simulation, we use iGibson-based environments [33]. The iGibson code is released under an open-source license, while the associated scene and asset datasets are governed by the corresponding iGibson dataset license agreement. We use these assets only for simulator execution and automatic evaluation.

**Language models.** For open-weight model evaluation, we use Qwen3.5-9B [30] under its Apache License 2.0. Gemma-3-12B [39] is used under the Gemma Terms of Use. For API-based proprietary or hosted models, including GPT-family, Gemini-family, DeepSeek-family, and MiMo-family models, we access the models through the corresponding official providers or API routing services available at evaluation time. Their use is governed by the relevant service terms, model terms, and usage policies of the corresponding providers. We do not redistribute proprietary model weights or API outputs as standalone model assets.

**Software and implementation dependencies.** Our implementation relies on standard open-source software packages for model inference, simulator control, data processing, and evaluation. We follow the licenses of these dependencies and will provide a dependency list with version information in the released code. All released benchmark annotations and task metadata will be distributed with a license compatible with the licenses and terms of the underlying simulators and assets.