# DEMANDAGENT: TOWARDS AN EMBODIED AGENT HARNESS FOR EVERYDAY HUMAN DEMANDS

**Anonymous authors**
Paper under double-blind review




Figure 1: **Demonstration of DEMANDAGENT**. An embodied agent harness for everyday assistance through demand understanding and navigation execution.

## ABSTRACT

Real-world embodied assistance often begins with a high-level human demand rather than a fully specified task. Existing embodied navigation settings typically ask agents either to reach a specified object goal or to navigate by following detailed route instructions. We argue that satisfying real-world human demands goes beyond localizing relevant targets and instead requires transforming high-level demands into executable embodied behaviors while considering inherent task dependencies. We introduce DEMANDNAV, a dependency-aware task formulation that represents a human demand as functional requirements and their prerequisite relations. To address this task, we present DEMANDAGENT, an embodied agent harness built around dedicated MCP tools for embodied interaction and persistent task-state and spatial-state management, guided by specialized embodied agent skills. We further introduce a top-down data construction pipeline and instantiate DEMAND-BENCH with 200 episodes from distinct ProcTHOR scenes. On DEMAND-BENCH, DEMANDAGENT achieves a 52.5% success rate, substantially outperforming adapted navigation baselines with static LLM-based demand decomposition and goal translation. Further evaluation on the public EB-Navigation benchmark shows that an adapted DEMANDAGENT achieves the best performance among the evaluated baselines, with a 65.7% average success rate, demonstrating the cross-benchmark adaptability of the proposed embodied agent harness. Code and data will be made publicly available.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

# 1 INTRODUCTION

Embodied agents are designed to assist people in everyday environments through autonomous perception, reasoning, and action. As illustrated in Fig. 1, users typically express desired outcomes rather than fully specifying executable tasks in real-world human-robot interactions. For example, a user might say, "The kitchen is pretty dirty. Help me clean it up." To satisfy such demands, an embodied agent must infer dependency-aware executable tasks and explore the environment to ground them. This constitutes an important yet underexplored challenge for embodied assistance.

In this work, we formulate the DEMANDNAV task and study human demand realization through embodied navigation. We argue that real-world embodied assistance requires bridging the gap between high-level human demands and executable task plans, rather than merely executing predefined goals or locating explicit targets. The agent must first infer functional requirements from the demand, reason about their prerequisite relations, determine valid execution orders, and finally ground these requirements to feasible objects within the environment. For example, given the demand "I'm thirsty," the agent may infer functional requirements such as a container and a water source, determine that obtaining a container should precede accessing the water source, and then ground these requirements to specific objects such as a cup and a water dispenser in the scene. Although achieving human demands may involve manipulation, we focus on navigation as a first step toward demand realization.

Embodied navigation benchmarks have gradually explored more complex task settings and less explicit goal specifications, including object-goal navigation (Anderson et al., 2018), multi-object navigation (Wani et al., 2020), and demand-driven navigation (Wang et al., 2023). Despite this progression, current benchmarks still oversimplify a central challenge of real-world embodied assistance. They either assume predefined goals and execution orders, or reduce demand realization to localizing relevant objects without modeling dependencies. As a result, the problem of inferring functional requirements and their prerequisite relations from human demands, while dynamically grounding them to concrete object instances in unknown scenes remains largely underexplored.

Conventional modular navigation frameworks (Zhang et al., 2025; Yin et al., 2024) and learned Vision-Language-Navigation (VLN) models (Zhang et al., 2024; Wei et al., 2026) typically operate under fixed targets or explicit instructions. Recent agentic navigation methods (Chandaka et al., 2025; Li et al., 2026b) offer a promising alternative through explicit reasoning and tool use, but still largely assume that the task objective is specified in advance.

In this paper, we present DEMANDAGENT, an embodied agent harness for everyday human demands. As illustrated in Fig. 2, DEMANDAGENT explicitly maintains inferred functional requirements, their prerequisite relations, and execution states within a closed-loop embodied decision process. We design dedicated MCP tools for environment interaction and persistent task-state and spatial-state management. Specifically, the navigation tools enable point-based visual navigation, while Adaptive Demand Ledger tools track task progress and support plan revision, and Spatial Memory tools preserve and retrieve spatial evidence to support later target and area revisitation.

We propose DEMAND-BENCH to encourage a shift toward real-world DEMANDNAV. Our benchmark follows a top-down, demand-first construction and contains 200 runtime-validated episodes across different ProcTHOR scenes in AI2-THOR (Kolve et al., 2017; Deitke et al., 2022). The evaluator accepts any admissible object assignment and any completion order consistent with the reference DAG. On DEMAND-BENCH, our DEMANDAGENT achieves a 52.5% success rate, substantially outperforming adapted navigation baselines with static LLM-based demand decomposition and goal translation. Beyond our benchmark, an adapted DEMANDAGENT achieves the best performance among the evaluated baselines on the public EB-Navigation benchmark, with a 65.7% average success rate, showing that the proposed harness remains effective under a different navigation setting.

In summary, our key contributions are as follows:

- **Task Formulation.** We introduce DEMANDNAV to shift embodied navigation from relevant target localization toward dependency-aware demand realization, requiring the agent to infer functional requirements and their prerequisite relations, and ground them to objects within the environment.
- **Embodied Agent Harness.** We present DEMANDAGENT, an embodied agent harness that leverages MCP tools for embodied interaction and persistent task-state and spatial-state management, guided by embodied agent skills for closed-loop reasoning and execution.

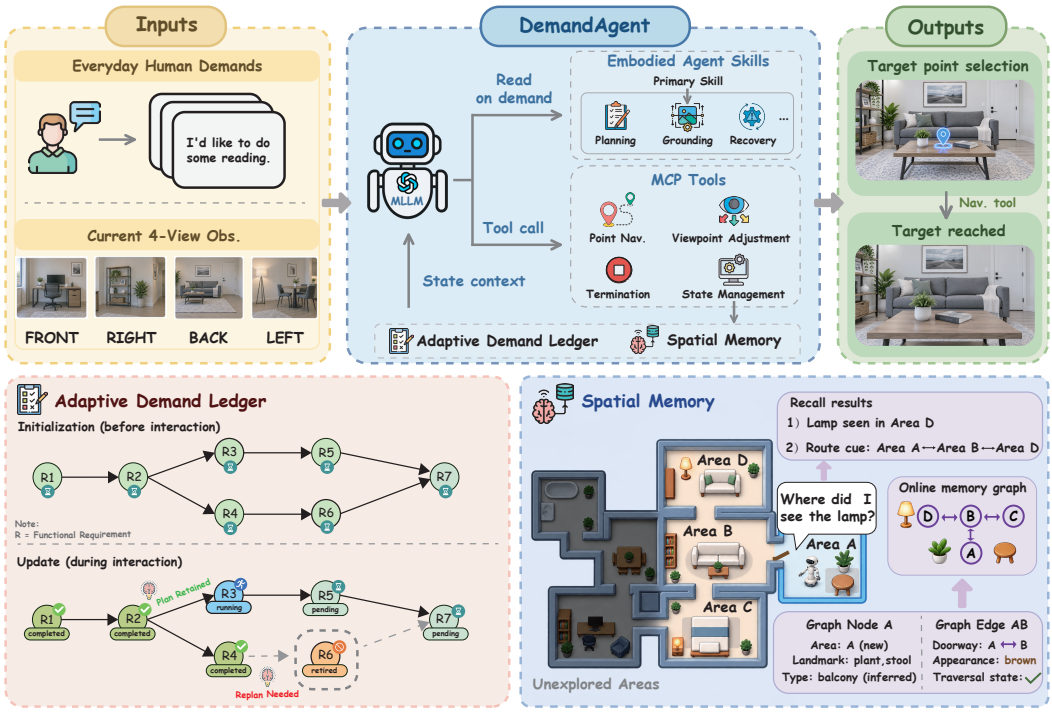


Figure 2: **Overview of DEMANDAGENT.** Top: given a human demand and current four-view RGB observations, DEMANDAGENT reads embodied agent skills and calls MCP tools to select the next navigation target, with execution results and state context fed back for subsequent decisions. Bottom: the Adaptive Demand Ledger tools manage task state and enable online replanning, while Spatial Memory tools incrementally update the memory graph and recall return route cues for revisitation.

- **Benchmark and Evaluation.** We construct DEMAND-BENCH through a top-down, demand-first procedure with 200 episodes across different ProcTHOR scenes and a multi-solution evaluator. Experiments demonstrate the effectiveness of DEMANDAGENT on DEMAND-BENCH and further evaluate its cross-benchmark adaptability on the public EB-Navigation benchmark.

## 2 RELATED WORK

### 2.1 DEMAND-DRIVEN EMBODIED NAVIGATION

Demand-driven navigation requires agents to infer relevant targets from human demands. DDN learns demand-conditioned object attributes to identify suitable resources (Wang et al., 2023), while IntentionNav evaluates object navigation from implicit human instructions (Qian et al., 2026). MO-DDN extends DDN to multiple objects and accepts alternative object combinations that satisfy a demand and its preferences (Wang et al., 2024). TP-MDDN considers instructions containing several expressed sub-demands and preferences. Its AWMSystem combines instruction decomposition, online target selection, and execution monitoring (Li et al., 2026a). These works establish demand interpretation, target selection, and multi-stage execution as important navigation capabilities.

Demand-oriented instruction generation is also relevant to our benchmark construction. NavRAG generates user-demand instructions by retrieving information from hierarchical scene descriptions, with simulated user roles and their specific needs guiding retrieval and generation (Wang et al., 2025).

Building on these settings, DEMANDAGENT explicitly models prerequisite relations among demand-inferred functional requirements and evaluates whether the agent grounds and completes these requirements in dependency-consistent orders.

### 2.2 Closed-Loop Planning and Embodied Navigation Agents

LLM-driven embodied robotics has explored diverse paradigms to align high-level cognitive planning with environmental feedback. SayCan combines language-based skill selection with affordance estimates (Ahn et al., 2022). Inner Monologue uses execution and scene feedback for closed-loop planning (Huang et al., 2022), while LLM-Planner generates and revises plans using observed objects (Song et al., 2023). SayPlan combines hierarchical scene-graph search with iterative plan correction (Rana et al., 2023). Navigation methods also adapt high-level decisions during exploration. SayNav updates search plans over an incrementally constructed scene graph for supplied object goals (Rajvanshi et al., 2024). InstructNav supports demand-driven instructions through a dynamically updated chain of navigation decisions (Long et al., 2024). CogDDN combines demand matching with heuristic and analytical reasoning (Huang et al., 2025). Together, these approaches highlight that online planning and demand understanding are essential for robust closed-loop navigation.

Recent work explicitly studies the infrastructure surrounding the reasoning model. ReasonNav and AgentVLN organize navigation around higher-level skills and modular reasoning (Chandaka et al., 2025; Xin et al., 2026). AgenticNav formulates continuous vision-and-language navigation as tool calling, with pixel-level movement targets, on-demand depth queries, and selective recall of past RGB observations indexed by a trajectory map (Li et al., 2026b). Thea transfers the coding-agent harness paradigm to physical robots through callable tools, procedural skills, task notes, persistent scene context, and action-outcome evaluation (Wang et al., 2026a). More broadly, a recent agentic navigation system combines tool use and skill guidance to support general-purpose embodied navigation (Anonymous, 2027). Building on these agentic designs, DemandAgent couples an explicit ledger of demand-inferred functional requirements, prerequisite relations, and execution states with area-level spatial memory for evidence retrieval and later revisitation.

## 3 Task Formulation

Real-world human demands naturally involve dependencies among functional requirements. DemandNav explicitly models the dependency structure of a demand $d$ using a directed acyclic graph (DAG) $G_d = (V_d, \mathcal{E}_d)$, where each node $v \in V_d$ denotes a functional requirement and each edge $(u, v) \in \mathcal{E}_d$ represents a prerequisite relation indicating that requirement $u$ must be completed before requirement $v$. A valid completion order is any topological ordering of $G_d$ that includes all functional requirements while respecting their prerequisite relations. Since multiple concrete objects may satisfy the same functional requirement and a DAG may allow multiple valid completion orders, task success cannot be tied to a single reference sequence. We therefore define the scene-conditioned feasible solution space as

$$\mathcal{R}(d, \mathcal{H}) = \{(b, \pi) \mid b \in \mathcal{B}(V_d, \mathcal{H}),\ \pi \in \Pi(G_d)\}, \tag{1}$$

where $\mathcal{B}(V_d, \mathcal{H})$ contains all valid assignments $b$ of functional requirements to object instances in scene $\mathcal{H}$, and $\Pi(G_d)$ contains all completion orders $\pi$ consistent with $G_d$.

The DemandNav task therefore requires the agent to infer a functional DAG from the human demand and ground its functional requirements to concrete objects in the scene through navigation.

## 4 DemandAgent

### 4.1 System Overview

To solve the DemandNav task, we implement DemandAgent as an embodied agent harness built on OpenAI Codex CLI (OpenAI, 2025). As illustrated in Figure 2, DemandAgent integrates a Model Context Protocol (MCP) tool interface (Section 4.2) with embodied agent skill guidance (Section 4.3). The MCP interface provides the agent with executable environment and state-management capabilities, while the skills provide procedural guidance for using these capabilities throughout execution. Throughout execution, DemandAgent maintains persistent task and spatial state through two complementary mechanisms. The Adaptive Demand Ledger (ADL) tracks task progress, while Spatial Memory preserves spatial state during interaction.

DemandAgent operates in a closed perception–reasoning–action loop. Before interacting with the environment, the agent infers an initial functional DAG from the human demand and initializes

the ADL. During execution, the agent leverages skills to reason over the latest visual observations with the maintained task and spatial states, and call the corresponding MCP tool for the next action. Environment interaction tools return fresh visual observations, whereas state-management tools maintain task and spatial state without altering the environment. The resulting information is incorporated into the context of the next reasoning step, allowing the agent to update task progress, revise the remaining plan when necessary, and reuse previously acquired spatial information. The loop continues until the agent issues STOP or exhausts the maximum step budget.

### 4.2 MCP Tool Interface

The executable capabilities of DemandAgent are provided through the MCP interface, including environment interaction tools (Section 4.2.1) and state management tools (Section 4.2.2 and 4.2.3).

#### 4.2.1 Environment Interaction Tools

We provide a set of environment interaction tools for point-based visual navigation, viewpoint adjustment, and termination, enabling the agent to interact efficiently with the environment.

Specifically, the navigation tool accepts a 2D image point selected from one of the four current RGB views (FRONT, RIGHT, BACK, or LEFT) and projects it to a reachable 3D endpoint using geometry in AI2-THOR simulator (Kolve et al., 2017) to execute a collision-free path. After execution, the tool returns the action status and a fresh set of four-view RGB observations. For visual targets outside the default viewing horizon, the agent can call Look_Down and Look_Up tools to adjust the camera pitch by a fixed angle and obtain fresh four-view observations. Once the task is judged complete, the agent can call the termination tool. By separating high-level embodied decision making from low-level geometric execution, these tools allow the agent to focus its reasoning on semantically meaningful decisions without directly handling low-level motion control.

#### 4.2.2 Adaptive Demand Ledger

Since a human demand specifies a desired outcome rather than a complete executable task plan, the agent should first infer a functional DAG from the original human demand and continually maintain task state. To this end, we introduce a demand ledger as a persistent task-state representation that augments the static functional DAG with four possible dynamic execution states for each requirement, which are PENDING, RUNNING, COMPLETED, or RETIRED. We further implement Adaptive Demand Ledger (ADL) with two tools: ADL_Init and ADL_Update.

The agent can call ADL_Init to initialize a demand ledger from the inferred functional DAG and subsequently call ADL_Update during execution to modify requirement states from PENDING to RUNNING and then to COMPLETED. The ADL_Update tool further provides a Replan operation for selectively revising the agent's task hypothesis with new evidence, by retiring a requirement, removing its associated edges from the active demand ledger, and marking it as RETIRED.

Overall, the ADL manages a persistent task state and explicitly maintains inferred functional requirements, prerequisite relations, and their evolving execution states throughout the episode.

#### 4.2.3 Spatial Memory

For DemandNav tasks, the agent may observe an object or area relevant to a later functional requirement before its prerequisites are satisfied. Without retaining this evidence, the agent may need to re-explore the environment later, potentially requiring repeated search. We therefore introduce Spatial Memory as a set of tools to preserve such spatial evidence for later retrieval.

The spatial memory is organized as an area-level graph, where each node represents an explored area with its associated landmarks, and each edge represents a doorway connecting two neighboring areas. Landmarks are stable objects recognized and recorded by the agent within an area. Spatial Memory is accessed through two tools: Memory_Update and Memory_Recall. With Memory_Update, the agent can write textual descriptions of areas, landmarks, and doorways, which are incrementally organized into the memory graph. Area descriptions record the agent-inferred area type and exploration status. Landmark descriptions capture task-relevant objects, including their appearance, inferred category, and relative spatial relations. Doorway descriptions record visual

appearance, traversal status, and visual cues about the connected areas. These records provide an explicit spatial state for reusing earlier observed spatial evidence.

Given a textual query about a target landmark or area, MEMORY_RECALL tool retrieves relevant records from the memory graph. At the object level, it performs lexical matching against stored landmark descriptions to identify the area where a relevant target was previously observed. At the area level, it identifies a relevant explored area and performs BFS over the memory graph to return a topological route as an ordered sequence of areas and connecting doorways.

The design of Spatial Memory preserves task-relevant spatial evidence across functional requirements, supporting later target and area revisitation.

### 4.3 EMBODIED AGENT SKILLS

Effective embodied execution requires procedural guidance for reasoning, interacting with the environment, and deciding when and how to call available tools. We therefore organize the embodied agent skills following a progressive-disclosure pattern (Chen et al., 2026; Xu & Yan, 2026). A primary skill remains available throughout the episode, while reference skills for ADL, Spatial Memory, grounding, and recovery are read on demand as they become relevant to the current operation.

Specifically, the ADL skill guides the agent to initialize the demand ledger with the ADL_INIT tool before environment interaction, and to call the ADL_UPDATE tool during execution to modify a requirement state from PENDING to RUNNING when selected as the current focus, and to COMPLETED only after visually confirming a valid target. The ADL skill further guides the agent to reassess the demand ledger before activating the next functional requirement and decide whether to retain it or invoke REPLAN operation based on the updated task state and recent observations. For example, for the demand "I'm thirsty and want some hot water," the inferred functional DAG may include requirements for a container, a water source, and heating equipment. If the agent later observes a water dispenser that directly provides hot drinking water, it can invoke REPLAN to retire the heating-equipment requirement and revise the demand ledger accordingly.

Moreover, the Spatial Memory skill guides the agent to record spatial evidence that may be useful for later search or revisitation, such as newly explored areas, traversed doorways, and task-relevant landmarks. It also guides the agent to call the MEMORY_RECALL tool when previously observed evidence becomes relevant to the current requirement or when local search makes little progress. After recall, the agent uses the returned area and doorway descriptions to re-identify the route from current RGB observations. If a recalled doorway is visually ambiguous, the skill guides the agent to obtain closer observations and compare them with the stored descriptions before proceeding.

Overall, the embodied agent skills provide structured guidance for tool use, environment interaction, and state-aware decision making, supporting coherent closed-loop execution throughout the task.

## 5 BENCHMARK CONSTRUCTION

DEMAND-BENCH follows a top-down construction pipeline. We manually define and review scene-independent demand templates across six everyday categories, specifying desired outcomes, functional DAGs, and admissible object types. Each template is grounded in ProcTHOR-10K scenes that support all its functional requirements (Deitke et al., 2022), followed by AI2-THOR validation of target reachability, visibility, and navigation feasibility. Each episode specifies one functional DAG. Solutions vary in admissible object assignments and DAG-consistent completion orders.

We generate demand instructions from desired outcomes using an LLM without exposing the functional DAG or scene-specific object information, preventing solution leakage into the instruction. After automated checks and manual review, DEMAND-BENCH contains 200 episodes across four difficulty levels. Construction details and dataset statistics are provided in Appendix A.

## 6 EXPERIMENTS

We evaluate DEMANDAGENT on four aspects: (1) How well can existing methods solve the DEMAND-NAV task under human-demand and oracle object-goal inputs? (2) How do individual components

contribute to demand satisfaction and execution efficiency? (3) How does performance change with task complexity, and what challenges remain? (4) How can DEMANDAGENT transfer to a public embodied navigation benchmark? We investigate these questions through comparative experiments, ablation studies, and further analyses. More details are provided in Appendix D.

### 6.1 EXPERIMENTAL SETUP

We evaluate DEMANDNAV on the 200-episode DEMAND-BENCH from distinct ProcTHOR scenes, while evaluation on the EB-Navigation benchmark follows its official protocol (Yang et al., 2025).

**Implementation details.** All experiments on DEMAND-BENCH use AI2-THOR 5.0.0. DEMANDAGENT uses GPT-5.6-Luna with max reasoning effort through Codex CLI 0.153.4. The MLLM receives only RGB observations, and each episode is limited to 500 MCP tool calls and 200 navigation decisions. An episode succeeds only if the agent issues STOP after completing all functional requirements in a DAG-consistent order. Additional details are provided in Appendix D.1.

**Evaluation metrics.** We report five primary metrics, following the standard SR/SPL (Anderson et al., 2018) convention while extending evaluation to dependency-aware demand realization and partial progress. **Success Rate (SR)** measures strict full-task success; **Requirement Coverage Rate (RCR)** measures valid functional requirement coverage while ignoring dependency order; and **Partial Success Rate (PSR)** credits only dependency-consistent partial progress. **Success weighted by Path Length (SPL)** measures the efficiency of complete demand realization, while **Partial Success weighted by Path Length (PSPL)** extends this efficiency measure to dependency-consistent partial progress. Detailed mathematical definitions and additional metrics are provided in Appendix C.

### 6.2 COMPARISON ON DEMAND-BENCH

Table 1 compares DEMANDAGENT with existing baseline methods on DEMANDNAV under two task-input settings. Since these baselines were originally implemented in Habitat-Sim (Savva et al., 2019), we adapt simulator-dependent components to AI2-THOR while retaining method-specific observation and action interfaces. In the **Human Demand** setting, each method is provided with the original demand. We use shared LLM adapters to convert the demand into either sub-demands or object goals before navigation, following their original input formats. In the **Oracle Object Sequence** setting, each method receives the ordered object goals from a shortest reference solution, while target instances, locations, and reference routes remain hidden. Details are provided in Appendix D.2.

LLM-adapted ObjectNav methods remain weak under the **Human Demand** setting. Fig. 3(b) illustrates a representative failure caused by static LLM-based demand translation, whereas DEMANDAGENT succeeds in Fig. 3(a), with additional successful cases shown in Fig. 3(d–f). The improvement with **Oracle Object Sequence** suggests that target selection and ordering errors contribute substantially to the performance gap. Results on additional metrics are provided in Appendix D.3.

### 6.3 ABLATION STUDY

Table 2 compares DEMANDAGENT with Vanilla Codex and variants of its visual navigation interface, ADL, Spatial Memory, and skills. Detailed variant definitions are provided in Appendix D.4.

DEMANDAGENT achieves the highest SR of 52.5% and PSR of 60.7%, compared with 31.5% SR for Vanilla Codex. Removing ADL, Spatial Memory, or Skills, or replacing the point-based navigation tool with the box-based alternative degrades overall performance. Fig. 3(a,d,e) shows memory-guided revisitation. Fig. 3(d) demonstrates replanning with new evidence. The retired heating-equipment requirement is an agent-inferred extra requirement absent from the reference DAG. REPLAN revises only the agent's task hypothesis, leaving the reference DAG unchanged for evaluation.

### 6.4 TASK COMPLEXITY AND REMAINING CHALLENGES

As shown in Fig. 4(a–b), DEMANDAGENT degrades substantially as task complexity increases: PSR drops from 90.0% on single-requirement tasks to 24.0% with four or more requirements, and to 21.2% for DAG depth 3 or greater. Missing requirements account for 56.8% of failures (Fig. 4(c)), making them a major failure source, as illustrated by the qualitative case in Fig. 3(c).

Table 1: **Comparison on DEMAND-BENCH.** Results under Human Demand and Oracle Object Sequence inputs, reported in %. The top-3 results are highlighted as first , second , and third .

| Method | Human Demand | | | | | Oracle Object Sequence | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | SR ↑ | RCR ↑ | PSR ↑ | SPL ↑ | PSPL ↑ | SR ↑ | RCR ↑ | PSR ↑ | SPL ↑ | PSPL ↑ |
| OpenFMNav (Kuang et al., 2024) | 5.0 | 18.2 | 15.2 | 4.5 | 8.9 | 8.0 | 26.0 | 17.0 | 7.2 | 11.5 |
| InstructNav (Long et al., 2024) | 5.5 | 21.0 | 17.4 | 5.1 | 9.6 | 9.0 | 28.0 | 18.2 | 8.5 | 12.0 |
| SG-Nav (Yin et al., 2024) | 4.5 | 18.5 | 13.2 | 4.1 | 8.1 | 7.0 | 24.5 | 15.6 | 6.2 | 10.6 |
| Uni-NaVid (Zhang et al., 2024) | 2.5 | 16.0 | 11.5 | 1.6 | 6.4 | 5.0 | 20.0 | 12.0 | 4.3 | 9.2 |
| ApexNav (Zhang et al., 2025) | 10.0 | 24.8 | 21.9 | 7.4 | 11.8 | 32.5 | 53.7 | 44.8 | 21.1 | 24.6 |
| WMNav (Nie et al., 2025) | 6.0 | 22.4 | 18.7 | 5.3 | 10.4 | 10.0 | 30.0 | 20.0 | 9.1 | 13.5 |
| CogNav (Cao et al., 2025) | 9.0 | 25.2 | 22.4 | 6.8 | 10.6 | 29.5 | 54.2 | 45.5 | 17.2 | 21.0 |
| **DEMANDAGENT (Ours)** | 52.5 | 61.1 | 60.7 | 26.1 | 28.4 | 76.0 | 84.4 | 83.8 | 41.4 | 42.9 |

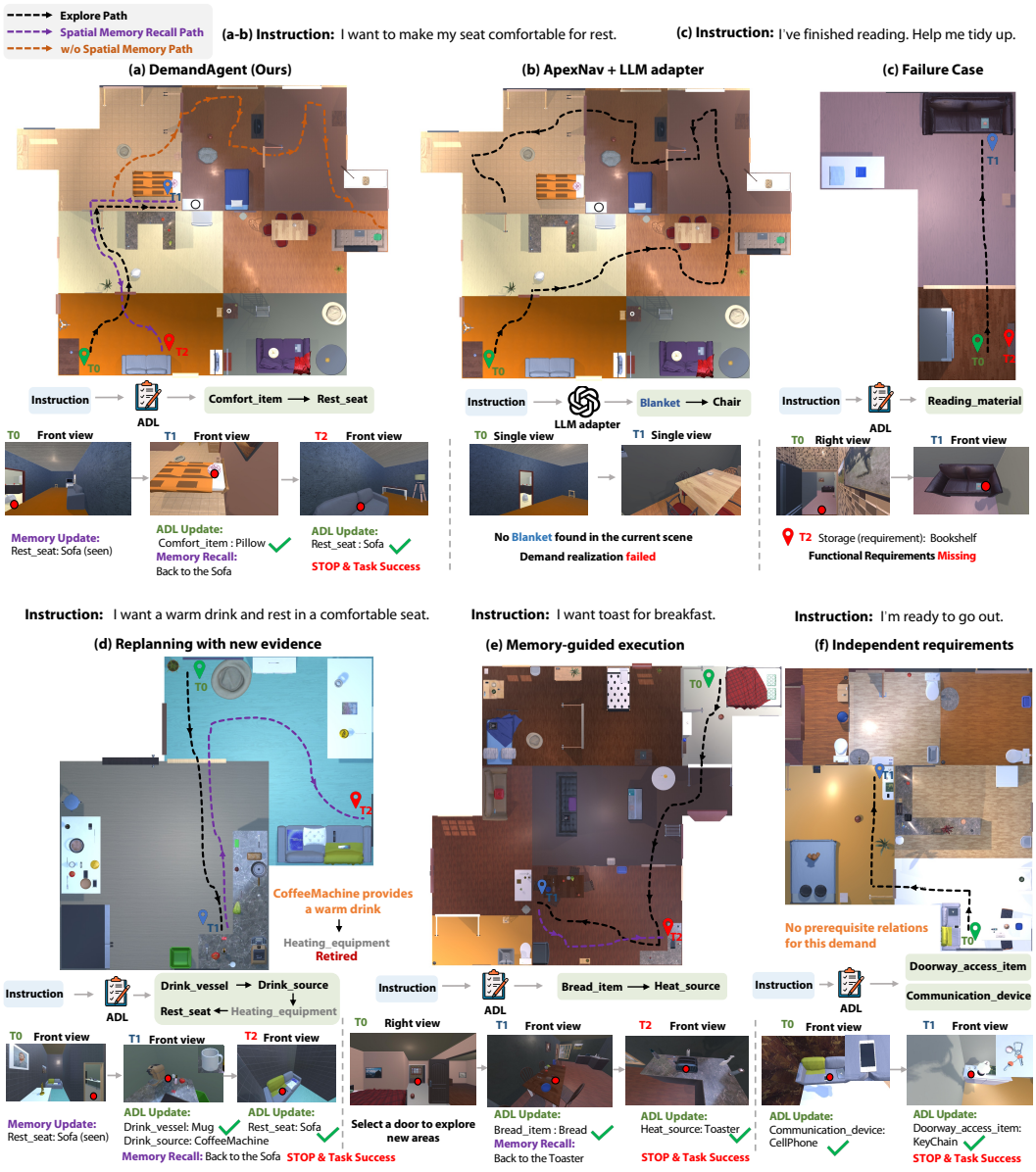


Figure 3: **Qualitative results on DEMAND-BENCH.** (a) Successful demand realization with memory recall. (b) Failure of ApexNav with LLM adapter. (c) Failure caused by a missing requirement. (d–f) Additional successful episodes involving replanning, memory recall, and independent requirements.

Table 2: **Ablation study on DEMAND-BENCH.** DEMANDAGENT and component variants.

| Variant | SR ↑ | RCR ↑ | PSR ↑ | SPL ↑ | PSPL ↑ |
|---|---|---|---|---|---|
| Vanilla Codex | 31.5 | 37.5 | 37.5 | 18.0 | 20.6 |
| w/o ADL | 42.0 | 54.6 | 54.6 | 21.3 | 25.1 |
| ADL w/o Replan | 51.5 | 60.2 | 60.2 | 25.5 | 28.0 |
| w/o Spatial Memory | 37.5 | 46.1 | 46.1 | 20.6 | 24.3 |
| w/o Skills | 34.0 | 42.7 | 42.7 | 19.5 | 23.0 |
| Box-based Nav. | 39.0 | 63.0 | 54.2 | 16.5 | 21.3 |
| **DEMANDAGENT (Ours)** | 52.5 | 61.1 | 60.7 | 26.1 | 28.4 |

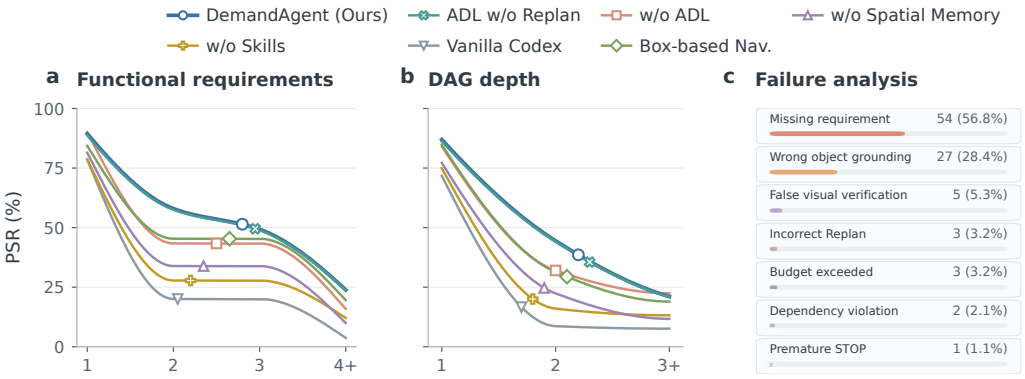


Figure 4: **Task complexity and failure analysis.** (a-b) PSR under increasing functional requirements and DAG depths. (c) Statistics of different failure types.

Table 3: **EB-Navigation SR (%).** "DEMANDAGENT (adapted)" follows the benchmark's interface.

| Method | Avg | Base | Common | Complex | Visual | Long |
|---|---|---|---|---|---|---|
| GPT-4o (Yang et al., 2025) | 57.7 | 55.0 | 60.0 | 58.3 | 60.0 | 55.0 |
| GPT-5.6-Luna | 59.3 | 68.3 | 66.7 | 68.3 | 60.0 | 33.3 |
| **DEMANDAGENT (adapted)** | 65.7 | 71.7 | 66.7 | 70.0 | 60.0 | 60.0 |

### 6.5 PUBLIC BENCHMARK EVALUATION

We adapt DEMANDAGENT to EB-Navigation's native interface and evaluate it under the official protocol. Both GPT-5.6-Luna and DEMANDAGENT use GPT-5.6-Luna with max reasoning effort. As shown in Table 3, DEMANDAGENT achieves the best overall performance, demonstrating its effectiveness on a different navigation benchmark. Details are provided in Appendix D.5.

## 7 LIMITATIONS AND FUTURE WORK

Despite the promising results, DEMANDAGENT currently focuses on embodied navigation, while many real-world demands also require physical manipulation. Extending the harness with manipulation capabilities could enable broader demand realization through physical interaction. In addition, Spatial Memory is currently limited to episode-level retention. Future work could explore lifelong memory and continual learning to enable more proactive, personalized embodied assistance.

## 8 CONCLUSION

We introduce DEMANDNAV, a dependency-aware task for human demand realization, and construct DEMAND-BENCH for evaluation. Our DEMANDAGENT supports closed-loop execution through dedicated navigation tools, the ADL, Spatial Memory, and embodied agent skills. Experiments validate the effectiveness of DEMANDAGENT and its adaptability to another navigation benchmark. This work takes a step toward embodied navigation agents that can autonomously transform high-level human demands into dependency-aware, scene-grounded, and closed-loop executable behaviors.

## AI USE STATEMENT

In this work, generative AI was used as part of the research and evaluation pipeline. During benchmark construction, a language model generated demand-instruction paraphrases, which underwent the automated semantic-frame checks and manual review described in Section 5. GPT-5.6-Luna, accessed through Codex CLI, also served as the semantic controller in the DEMANDAGENT and EB-Navigation evaluations and is therefore part of the evaluated system; Sections 4 and 6 describe its information boundary, evaluation protocol, and results. We did not use generative AI for other required-disclosure tasks; tasks involving mathematical claims, proofs, or human subjects were not applicable.

Beyond these research components, we used generative AI only to polish the manuscript for clarity and readability. The authors reviewed all AI-assisted text and verified the final claims and descriptions. We take responsibility for the final content of this work, including text produced with the aid of generative AI.

## REFERENCES

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.

Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.

Anonymous. Supernav: An agentic navigation system for any task in any scene, 2027. Manuscript under review at ICLR 2027.

Yihan Cao, Jiazhao Zhang, Zhinan Yu, Shuzhen Liu, Zheng Qin, Qin Zou, Bo Du, and Kai Xu. Cognav: Cognitive process modeling for object goal navigation with llms. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9550–9560. IEEE, 2025.

Bhargav Chandaka, Gloria X Wang, Haozhe Chen, Henry Che, Albert J Zhai, and Shenlong Wang. Human-like navigation in a world built for humans. *arXiv preprint arXiv:2509.21189*, 2025.

Zhiyu Chen, Zihan Guo, Bo Huang, Bingwei Lu, Jianghao Lin, Yuanjian Zhou, and Weinan Zhang. Skilljuror: Measuring how agent skill organization changes runtime behavior. *arXiv preprint arXiv:2606.11543*, 2026.

Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Kiana Ehsani, Jordi Salvador, Winson Han, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. Procthor: Large-scale embodied ai using procedural generation. *Advances in neural information processing systems*, 35:5982–5994, 2022.

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.

Yuehao Huang, Liang Liu, Shuangming Lei, Yukai Ma, Hao Su, Jianbiao Mei, Pengxiang Zhao, Yaqing Gu, Yong Liu, and Jiajun Lv. Cogddn: A cognitive demand-driven navigation with decision optimization and dual-process thinking. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 5237–5246, 2025.

Mukul Khanna, Ram Ramrakhya, Gunjan Chhablani, Sriram Yenamandra, Theophile Gervet, Matthew Chang, Zsolt Kira, Devendra Singh Chaplot, Dhruv Batra, and Roozbeh Mottaghi. Goat-bench: A benchmark for multi-modal lifelong navigation. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16373–16383. IEEE, 2024.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, et al. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017.

Jacob Krantz, Erik Wijmans, Arjun Majumdar, Dhruv Batra, and Stefan Lee. Beyond the nav-graph: Vision-and-language navigation in continuous environments. In *European Conference on Computer Vision*, pp. 104–120. Springer, 2020.

Yuxuan Kuang, Hai Lin, and Meng Jiang. Openfmnav: Towards open-set zero-shot object navigation via vision-language foundation models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 338–351, 2024.

Shanshan Li, Da Huang, Yu He, Yanwei Fu, Yu-Gang Jiang, and Xiangyang Xue. Tp-mddn: Task-preferenced multi-demand-driven navigation with autonomous decision-making. *Advances in Neural Information Processing Systems*, 38:84135–84154, 2026a.

Yijian Li, Changze Li, Hantian Shi, Jiaying Luo, Jiyuan Cai, Ming Yang, and Tong Qin. Agenticnav: Zero-shot vision-and-language navigation as a tool-calling harness. *arXiv preprint arXiv:2606.10577*, 2026b.

Yuxing Long, Wenzhe Cai, Hongcheng Wang, Guanqi Zhan, and Hao Dong. Instructnav: Zero-shot system for generic instruction navigation in unexplored environment. *arXiv preprint arXiv:2406.04882*, 2024.

Dujun Nie, Xianda Guo, Yiqun Duan, Ruijun Zhang, and Long Chen. Wmnav: Integrating vision-language models into world models for object goal navigation. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2392–2399. IEEE, 2025.

OpenAI. Codex cli. `https://github.com/openai/codex`, 2025. Accessed: 2026-09-08.

Lin Qian, Shijie Li, Sihao Lin, Xuan Zhang, Bangya Liu, Yanran Li, and Hujun Yin. Intentionnav: A benchmark for intent-driven object navigation from implicit human instruction. *arXiv preprint arXiv:2605.23187*, 2026.

Abhinav Rajvanshi, Karan Sikka, Xiao Lin, Bhoram Lee, Han-Pang Chiu, and Alvaro Velasquez. Saynav: Grounding large language models for dynamic planning to navigation in new environments. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pp. 464–474, 2024.

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.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9338–9346. IEEE, 2019.

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*, pp. 2998–3009, 2023.

Hongcheng Wang, Andy Guan Hong Chen, Xiaoqi Li, Mingdong Wu, and Hao Dong. Find what you want: Learning demand-conditioned object attribute space for demand-driven navigation. *Advances in Neural Information Processing Systems*, 36:16353–16366, 2023.

Hongcheng Wang, Peiqi Liu, Wenzhe Cai, Mingdong Wu, Zhengyu Qian, and Hao Dong. Moddn: a coarse-to-fine attribute-based exploration agent for multi-object demand-driven navigation. *Advances in Neural Information Processing Systems*, 37:64176–64214, 2024.

Qi Wang, Tianyi Wang, Chengyang Li, Shikun Ban, Yurun Chen, Yizhong Ge, Jason Qin, Chengtai Li, and Wentao Zhu. Towards the harness of embodied agents. *arXiv preprint arXiv:2608.11246*, 2026a.

Shihao Wang, Shilong Liu, Yuanguo Kuang, Xinyu Wei, Yangzhou Liu, Zhiqi Li, Yunze Man, Guo Chen, Andrew Tao, Guilin Liu, et al. Locateanything: Fast and high-quality vision-language grounding with parallel box decoding. *arXiv preprint arXiv:2605.27365*, 2026b.

Zihan Wang, Yaohui Zhu, Gim Hee Lee, and Yachun Fan. Navrag: Generating user demand instructions for embodied navigation through retrieval-augmented llm. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 8430–8440, 2025.

Saim Wani, Shivansh Patel, Unnat Jain, Angel Chang, and Manolis Savva. Multion: Benchmarking semantic map memory using multi-object navigation. *Advances in Neural Information Processing Systems*, 33:9700–9712, 2020.

Meng Wei, Chenyang Wan, Peng Peng, Xiqian Yu, Yuqiang Yang, Delin Feng, Wenzhe Cai, Chenming Zhu, Tai Wang, Jiangmiao Pang, et al. Ground slow, move fast: A dual-system foundation model for generalizable vision-language navigation. In *International Conference on Learning Representations*, volume 2026, pp. 12380–12396, 2026.

Zihao Xin, Wentong Li, Yixuan Jiang, Ziyuan Huang, Bin Wang, Piji Li, Jianke Zhu, Jie Qin, and Shengjun Huang. Agentvln: Towards agentic vision-and-language navigation. *arXiv preprint arXiv:2603.17670*, 2026.

Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward. *arXiv preprint arXiv:2602.12430*, 2026.

Karmesh Yadav, Ram Ramrakhya, Santhosh Kumar Ramakrishnan, Theo Gervet, John Turner, Aaron Gokaslan, Noah Maestre, Angel Xuan Chang, Dhruv Batra, Manolis Savva, et al. Habitat-matterport 3d semantics dataset. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4927–4936. IEEE, 2023.

Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, et al. Embodiedbench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. *arXiv preprint arXiv:2502.09560*, 2025.

Hang Yin, Xiuwei Xu, Zhenyu Wu, Jie Zhou, and Jiwen Lu. Sg-nav: Online 3d scene graph prompting for llm-based zero-shot object navigation. *Advances in neural information processing systems*, 37: 5285–5307, 2024.

Naoki Yokoyama, Ram Ramrakhya, Abhishek Das, Dhruv Batra, and Sehoon Ha. Hm3d-ovon: A dataset and benchmark for open-vocabulary object goal navigation. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5543–5550. IEEE, 2024.

Jiazhao Zhang, Kunyu Wang, Shaoan Wang, Minghan Li, Haoran Liu, Songlin Wei, Zhongyuan Wang, Zhizheng Zhang, and He Wang. Uni-navid: A video-based vision-language-action model for unifying embodied navigation tasks. *arXiv preprint arXiv:2412.06224*, 2024.

Mingjie Zhang, Yuheng Du, Chengkai Wu, Jinni Zhou, Zhenchao Qi, Jun Ma, and Boyu Zhou. Apexnav: An adaptive exploration strategy for zero-shot object navigation with target-centric semantic fusion. *IEEE Robotics and Automation Letters*, 2025.

# Appendix

## A Benchmark Details

### A.1 Construction pipeline

DEMAND-BENCH follows the top-down construction pipeline described in Section 5. We manually define and review scene-independent *demand templates* across six everyday categories. Each template specifies a desired outcome, a functional DAG, and admissible object categories. The DAG encodes functional requirements and their prerequisites. These specifications define navigation and visual-verification tasks. Requests concerning cooking, cleaning, or storage do not require the agent to perform the corresponding physical manipulation.

We ground each template in ProcTHOR-10K homes that support its functional requirements. Static grounding matches requirements to object instances and checks required spatial relations. AI2-THOR simulator then validates target reachability, visibility, and navigation feasibility by checking candidate endpoints and connecting routes. Demand instructions are generated from the desired outcomes and undergo automated checks and manual review before episode assembly.

DEMAND-BENCH consists of 200 episodes in 200 distinct ProcTHOR-10K homes. Each episode uses a single functional DAG, stored as its reference annotation. Multiple solutions arise from alternative admissible object-binding assignments and DAG-consistent completion orders. The reference route provides a minimum-cost realization under a fixed endpoint and routing policy while its particular object-binding assignment and order do not prescribe a unique solution.

Table 4 compares the task-specification boundary across related embodied navigation benchmarks. In this comparison, *goal* follows each benchmark's native terminology. For DEMAND-BENCH, each inferred goal corresponds to a functional requirement, whereas an ObjectNav goal is an object-category target rather than a functional requirement. *Implicit Goal* means that goal identities, counts, or compositions are not explicitly specified and must be inferred from the task input. *Dependency-Aware* means that functional requirements and their prerequisite relations must be inferred and respected during execution; it does not refer to a target visitation order explicitly provided in the task input. In the Dependency-Aware column, — denotes tasks with only a single goal or for which inter-goal dependency is not applicable. The $\times$ symbol denotes multi-goal tasks whose formulation does not include prerequisite or dependency structure. The ✓ symbol is reserved for tasks that explicitly require the agent to reason about and respect prerequisite relations among inferred requirements or goals.

### A.2 Demand categories and instruction generation

Table 5 gives category counts and one example demand from each category. The final column lists the functional requirements for the example on the same row through their concise role descriptions. These requirements do not apply to every demand in that category, and the table omits their dependency edges and scene-specific object candidates.

Demand instructions are generated with GPT-5.5 at high reasoning effort. The initial writer receives desired-outcome information without the functional DAG, scene-specific objects, or reference solutions. We distinguish activity-explicit demands (D1) and intent-only demands (D2). D1 may state the resident's intended activities, while D2 states the desired outcome and authorized context without enumerating its component activities. For example, a D1 request says, "I'd like to make a warm drink and unwind in a comfortable seat while I have it," whereas a D2 request says, "I'm thirsty and want to unwind with some entertainment for a while." The benchmark contains 147 D1 and 53 D2 episodes. Automated validation has two stages. First, deterministic filters reject malformed candidates, formal planning terms, object identifiers, and execution-order phrases. They also compare each candidate's declared meaning with the content permitted by its demand template, including singular or plural cardinality, activities and final states, manipulation and spatial-relation claims, and whether any named household item may be stated. Second, a separate LLM reviewer checks that the instruction is natural spoken English, faithfully expresses the intended need, follows the assigned D1/D2 abstraction level, and avoids benchmark or ontology terminology. The resulting instructions

Table 4: **Comparison of embodied navigation benchmarks.** We compare task inputs, whether goals are implicit, and whether execution is dependency-aware.

| Benchmark | Task Input | Multi-Goal | Implicit Goal | Dependency-Aware |
|---|---|---|---|---|
| VLN-CE (R2R, RxR) (Krantz et al., 2020) | Route Instruction | × | — | — |
| HM3D ObjectNav (Yadav et al., 2023) | Object Category | × | × | — |
| HM3D-OVON (Yokoyama et al., 2024) | Free-form Object Goal | × | × | — |
| MultiON (Wani et al., 2020) | Object Sequence | ✓ | × | × |
| GOAT-Bench (Khanna et al., 2024) | Multi-modal Goal Sequence | ✓ | × | × |
| DDN (Wang et al., 2023) | Demand | × | ✓ | — |
| MO-DDN (Wang et al., 2024) | Demand + Preference | ✓ | ✓ | × |
| TP-MDDN (Li et al., 2026a) | Multi-Demand + Preference | ✓ | ✓ | × |
| IntentionNav (Qian et al., 2026) | Intent | × | ✓ | — |
| **DEMAND-BENCH (Ours)** | Demand | ✓ | ✓ | ✓ |

Table 5: Recorded demand-category assignments in DEMAND-BENCH. The final two columns show one example demand and its functional requirements per category.

| ID | Category | Count | Example demand | Functional requirements (role descriptions) |
|---|---|---|---|---|
| N1 | Bodily maintenance and daily routines | 59 | Heat a snack | food item, microwave-compatible container, microwave |
| N2 | Environmental maintenance and spatial order | 38 | Clean a countertop | cleaning tool, cleaning supply, kitchen surface |
| N3 | Safety checks and risk prevention | 10 | Clear floor obstacles | two distinct floor obstacles, storage location |
| N4 | Productivity, learning, and instrumental tasks | 23 | Clear a workspace | work surface, fragile item, stable display surface |
| N5 | Leisure and sensory comfort | 33 | Put out decorations | decorative item, decorative feature, stable display surface |
| N6 | Cognitive support and memory retrieval | 37 | Locate misplaced belongings | living-room surface, frequently misplaced item, personal item |

undergo the author review described in Section 5. Instruction mention order does not impose an execution constraint.

### A.3 TEMPLATE AND DEPENDENCY ANNOTATION REVIEW

We distinguish named template schemas from their functional structure. The final 200-episode manifest contains 83 named schemas and 82 distinct requirement–DAG signatures, where a signature is the exact set of role-described functional requirements and direct prerequisite edges, independent of scene-specific object instances. Two named schemas share one signature because they express different desired outcomes using the same two functional requirements. Table 6 reports the complete episode allocation over the 82 distinct structures.

Table 7 gives one reviewed example. The demand is "I'd like to make a warm drink and unwind in a comfortable seat while I have it." Each node denotes a function to be supported, not a fixed object identity, and each edge records a functional prerequisite rather than the order in which objects happen to be mentioned.

The authors reviewed template semantics before scene grounding and conducted a separate episode-level pass after materialization. For each of the 200 episodes, the review packet presented the demand, desired outcome, functional requirements, prerequisite map, admissible object categories, and representative legal bindings and completion orders. The review checked that every requirement served the desired outcome, every edge expressed a genuine functional prerequisite rather than a convenient visitation order, and the instruction neither omitted required meaning nor exposed the reference DAG. Revised entries were returned to the same checks before the benchmark was frozen. This was an author annotation audit, not a human-subject study: it was separate from automatic

Table 6: Distribution of the 200 episodes over distinct functional-requirement and prerequisite structures.

| Episodes per structure | Structures | Episodes |
|---|---|---|
| 1 | 43 | 43 |
| 2 | 21 | 42 |
| 3 | 5 | 15 |
| 4 | 4 | 16 |
| 6 | 2 | 12 |
| 7 | 1 | 7 |
| 8 | 3 | 24 |
| 12 | 1 | 12 |
| 13 | 1 | 13 |
| 16 | 1 | 16 |
| **Total** | **82** | **200** |

Table 7: Representative demand–DAG annotation and the rationale for each direct prerequisite. Object categories illustrate admissible bindings.

| Functional requirement | Role in the demand | Example category | Direct prerequisite and rationale |
|---|---|---|---|
| Drink vessel | Provides a suitable container for the beverage | Mug | None |
| Coffee machine | Provides the equipment needed for a warm beverage | CoffeeMachine | Drink vessel, because a suitable container must be available before establishing the warm-beverage setup |
| Rest seat | Provides the comfortable place in the desired final state | Sofa | Coffee machine, because the warm-beverage setup precedes the final seated-consumption state |

instruction generation, but was not a blinded second-annotator study independent of all initial template authors, and we do not report inter-annotator agreement.

Ambiguity was resolved at the template boundary. Alternative object categories, instances, and completion orders were retained when they realized the same functional requirements under the same prerequisite graph. An interpretation that required a different requirement set or different edges was not counted as another solution to that episode. Instead, the wording or template was narrowed to a single defensible scope, and a candidate was excluded when that scope could not be established without an arbitrary semantic choice or without revealing the intended requirements or their order.

### A.4 Scene grounding and valid solution sets

Scene grounding begins by matching each demand template to scenes that contain valid candidates for all functional requirements. A functional requirement is a node in the reference DAG and is described by the function, or role, that must be fulfilled. An object category denotes an admissible type for that requirement, and an object instance is a particular scene entity. An object-binding assignment maps each functional requirement to one admissible object instance. We retain only assignments that use admissible object categories, assign distinct object instances to different functional requirements, and satisfy the required scene relations. All retained reference realizations respect the episode's designated prerequisite graph, while object-specific spatial relations additionally restrict binding validity. The reference graph is not inferred anew for each stored completion order. Runtime validation checks whether required objects are reachable and visible from valid agent poses and whether a feasible route connects them from the fixed start in a DAG-consistent order. An episode is retained only if at least one complete assignment and executable route passes these checks.

Each episode is defined by a reference DAG that specifies its functional requirements and prerequisite relations. Different valid solutions may bind these requirements to different admissible object instances and execute them in any DAG-consistent order from the same start state. The serialized requirement list is used only for storage and does not introduce additional ordering constraints.

Each compressed solution sidecar stores the requirement order, bound object instances, reference cost, and an optional schema-group identifier. We retain only entries from the episode's designated reference schema and verify that they cover all requirements in the reference DAG, use admissible object instances without reuse, and satisfy all prerequisite relations. After restricting each episode to its designated reference schema, the 200-episode set contains 43,380 valid realizations.

Across the 200 benchmark episodes, this process yields 43,380 DAG-consistent realizations. Exhaustiveness is defined within the designated schema, candidate, endpoint, and routing contracts rather than over all physically possible solutions. Full waypoint geometry is stored only for the minimum-cost realization, while the remaining realizations retain their logical specification and reference cost. Separate runtime-route records store endpoint indices and segment costs for reference reconstruction.

### A.5 Runtime endpoint and reference-route validation

The simulation setup uses a $120^\circ$ horizontal camera field of view, a 0.1 m navigation grid and movement resolution, and a simulator visibility range of 20 m. This visibility range is independent of the 1.5 m endpoint acceptance threshold. Object visibility alone does not constitute successful arrival or requirement completion. For ordinary objects, endpoint distance is measured horizontally to the center of the selected object. For eligible small objects placed on an exposed support, distance is instead measured to the horizontal footprint of the supporting surface, while the selected object must remain visible.

Endpoint selection starts from reachable positions within the target room, ranked by distance to the selected object center, and chooses the first pose that satisfies all runtime checks. The agent is oriented toward the object center, with camera horizons evaluated at $0^\circ$, $30^\circ$, and $60^\circ$. A valid endpoint must be interactable, exactly reachable by the simulator without forced correction, and collision-free, with the selected object remaining visible from that pose. Validation preserves the initial scene state, so targets that become visible only after opening, moving, or otherwise manipulating a receptacle are excluded. Once validated, the endpoint remains fixed for that object during route enumeration.

Within a room, the generator uses the native Unity NavMesh route if its polyline stays inside the room, checked at 0.1 m intervals, otherwise it uses a shortest cardinal path over reachable points in that room. Cross-room routes connect these paths through per-side anchors and the authoritative doorway centers. Length is the sum of horizontal $x$–$z$ waypoint distances in meters, while vertical displacement and camera rotation do not contribute. The designated reference attains the minimum cost among the retained DAG-consistent realizations. Its optimality is conditional on the fixed endpoints and doorway policy, rather than a continuous optimization over all possible stopping poses.

### A.6 Difficulty

The construction pipeline estimates difficulty using a heuristic score that combines the number of functional requirements, the spatial spread of candidate objects, prerequisite depth, DAG structure, and the presence of alternative assignments or completion orders:

$$s = \max(0, K-2) + \max(0, R-1) + \max(0, H-1) + T + A. \tag{2}$$

Here, $K$ is the number of functional requirements, $R$ is the number of distinct rooms containing their candidate objects, and $H$ is the number of functional requirements on the longest prerequisite chain, with $H = 1$ for an unordered DAG. The structural term $T$ is 0 for unordered DAGs, 1 for chains, 2 for forks or joins, and 3 for mixed DAGs or structures containing both branching and merging. The alternative term $A$ is 1 when there is more than one combination of candidate objects or more than one DAG-consistent completion order, and 0 otherwise. Both $R$ and $A$ are evaluated before simulator validation: they describe the candidate search space, rather than the rooms visited or the valid solutions retained in the final reference annotation.

The initial grading thresholds are Easy ($s \leq 2$), Medium ($3 \leq s \leq 5$), Hard ($6 \leq s \leq 8$), and Expert ($s \geq 9$). The released episodes retain their recorded difficulty labels through subsequent annotation repairs, so these thresholds describe the construction-stage heuristic and do not necessarily reproduce every final label from the revised DAG and candidate sets.

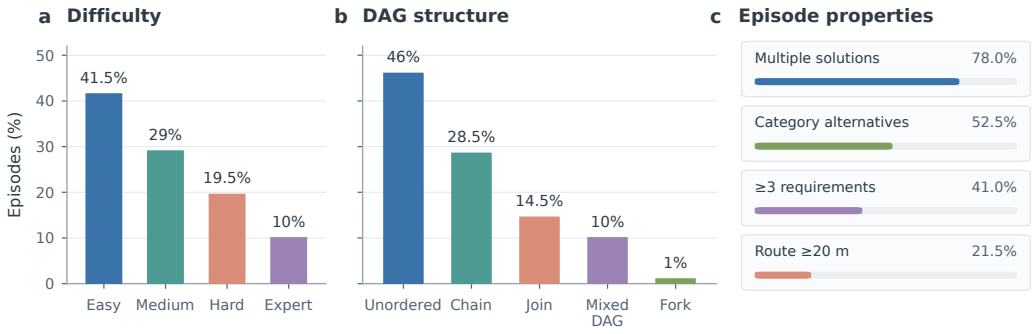


Figure 5: **Statistics of the 200-episode DEMAND-BENCH.** (a) Difficulty distribution. (b) Five DAG structures. (c) Solution diversity, number of functional requirements, and reference route length.

The benchmark spans four difficulty levels: Easy, Medium, Hard, and Expert, with 83, 58, 39, and 20 episodes, respectively. Hard or Expert episodes account for 29.50% of the benchmark. Figure 5 shows the difficulty distribution together with structural and solution-diversity statistics.

### A.7 BENCHMARK STATISTICS

Figure 5 summarizes the 200 benchmark episodes. Episodes contain a mean of 2.33 functional requirements, a median of 2, and a range of 1–8. We classify each reference DAG after removing prerequisite edges whose ordering is already implied by longer paths. Five structures occur: unordered DAGs have no prerequisite edges, chains order all functional requirements in one sequence, forks branch from one functional requirement to multiple successors, and joins require multiple predecessors before a functional requirement. Mixed DAGs cover the remaining combinations, including independent chains or ordered requirements alongside unconstrained requirements.

The four properties in Figure 5(c) describe solution diversity and task complexity. Multiple solutions means that the reference DAG admits at least two valid object-binding assignments or completion orders, whereas category alternatives means that its functional requirements admit more than one combination of object categories. The remaining two properties report episodes with at least three functional requirements and those with a reference route of at least 20 m.

## B TASK EXECUTION AND EVALUATION PROTOCOL

### B.1 INFORMATION BOUNDARY AND OBSERVATION TIMING

DEMANDAGENT maintains clear information boundaries among the MLLM, navigation executor, and offline evaluator. The MLLM receives the demand, the public tool interfaces, RGB observations, and records from ADL and Spatial Memory when those modules are enabled. It does not receive simulator object identifiers, object poses, depth, segmentation, reachable positions, or reference routes. The navigation executor privately uses depth, camera pose, and reachability information to convert an image point into collision-checked motion, returning only the execution result.

In the full ADL-enabled system, the agent first receives the human demand without visual observations. During this initialization stage, the agent can infer a functional DAG from the instruction and invoke ADL_INIT under the skill guidance. After the demand ledger is initialized, the initial FRONT, RIGHT, BACK, and LEFT observations are provided and normal environment interaction begins. The w/o ADL variant uses the same observation-release schedule, so disabling ADL removes the ledger rather than providing earlier visual access. When enabled, state-management tools update the agent's internal state without directly triggering movement or new visual observations.

### B.2 Completion Events

Observation, memory recall, navigation arrival, public requirement completion, and evaluator credit are distinct events. In the full system, the skill instructs the agent to submit a completion update based on fresh visual observations. A public update does not automatically receive evaluator credit. For scoring, a completion event is anchored to an explicit target-directed decision rather than to incidental object visibility. The public record identifies the intended functional requirement in semantic language and the visual target selected from the current observation, and the full system also records a natural-language binding and its fresh visual justification in the ADL. These records contain no simulator object identifiers.

On the private evaluation side, the selected visual target is resolved to the corresponding object instance and compared with the admissible bindings of the reference requirement. When several similar objects are visible, the selected image location (or selected proposal in the box-based variant) determines which object instance is evaluated, while the claimed function and reference relation constraints further restrict the match. The evaluator then checks target compatibility, distance, visibility, instance reuse, and predecessor credit independently of the public ledger. Private simulator metadata used for this matching is never exposed to the policy. Accidental co-visibility or an unexecuted semantic mention therefore produces no completion credit.

Here, an ObjectNav goal denotes the baseline's native target object category, not a DemandNav functional requirement. ADL operations are not required to enter this common event stream. For ObjectNav baselines, the evaluation wrapper emits the equivalent event when a target-directed action satisfies the current object-goal completion rule and the method advances to the next object goal. Vanilla Codex and the w/o ADL variant similarly produce an event when their target-directed action for the current intended requirement passes the completion checks. All such events are scored by the same private requirement–instance matching and validity checks. Omitting ADL changes the agent's task-state mechanism, not the evaluator's acceptance criteria. Early visibility does not earn predecessor-consistent credit; the scorer needs a valid completion record after predecessor credit exists.

### B.3 Episode Termination and Budgets

In the full system, STOP is an agent-issued terminal action. Final episode success is determined independently by the offline evaluator without exposing corrective target information to the agent. Reaching the tool-call or navigation limit terminates the episode automatically rather than through an agent-issued STOP, and therefore does not count as episode success.

### B.4 Evaluation Semantics and Oracle Setting

The evaluator uses the fixed reference DAG defined by each episode, while the ADL represents the agent's current task hypothesis. Accordingly, Replan can revise only the agent-side demand ledger and cannot alter the requirements or dependencies used for evaluation. The agent may therefore complete its revised ledger while still failing the offline evaluation.

Under **Human Demand** setting, completion is matched against all admissible bindings defined for the episode. Under **Oracle Object Sequence** setting, the agent receives one valid ordered sequence of object categories, while target object instances, locations, and reference routes remain hidden.

## C Evaluation Metrics

### C.1 Primary Metrics

For each of $N$ evaluation episodes, let $V_i$ be the set of required functional requirements. Let $S_i = 1$ only when all requirements in $V_i$ receive legal credit and the policy terminates with a valid STOP. Success Rate is

$$\mathrm{SR} = \frac{1}{N} \sum_{i=1}^{N} S_i. \tag{3}$$

SR is deliberately strict: it jointly tests whether the agent completes the entire demand with valid objects, respects the dependency structure, and terminates correctly. It therefore captures full demand satisfaction under the complete evaluation protocol, but does not distinguish unsuccessful episodes that make different amounts of valid progress.

To measure partial progress, let $w_{iv} > 0$ be the fixed weight of requirement $v$ (uniform in the reported evaluation), and let $W_i = \sum_{v \in V_i} w_{iv}$. Let $y_{iv} = 1$ when an explicit completion event for $v$ passes the evaluator's binding, relation, distance, visibility, and instance-reuse checks, irrespective of dependency order. Let $c_{iv} = 1$ only when the same event occurs after all predecessors of $v$ have already received dependency-consistent credit. The episode-level Requirement Coverage Rate (RCR) and Partial Success Rate (PSR) are

$$\mathrm{RCR}_i = \frac{1}{W_i} \sum_{v \in V_i} w_{iv} y_{iv}, \qquad \mathrm{PSR}_i = q_i = \frac{1}{W_i} \sum_{v \in V_i} w_{iv} c_{iv}. \tag{4}$$

Thus, RCR relaxes only dependency order and does not relax the validity of a binding or completion event. PSR measures dependency-consistent partial progress. RCR isolates how much valid functional coverage the agent discovers, even when the execution order is illegal, whereas PSR restores the task's prerequisite structure. Their difference therefore exposes progress lost to ordering errors, and both complement the all-or-nothing SR with graded measures of incomplete demand realization.

For path efficiency, $P_i$ is the translation distance accumulated from successful primitive motion in the evaluated trajectory. Camera rotations, failed primitives, and state-management calls add no translation, while all successful translation through termination or episode end is retained. Let $\mathcal{R}_i^{\mathrm{ref}}$ denote the benchmark's validated complete reference realizations, including alternative admissible bindings and legal orders, and let $L_i(r)$ be the corresponding executable route length. Define

$$L_i^* = \min_{r \in \mathcal{R}_i^{\mathrm{ref}}} L_i(r). \tag{5}$$

Partial references are constructed from the validated complete realizations. For each realization $r = ((v_1, b_1), \ldots, (v_K, b_K))$, the evaluator considers every nonempty prefix $r_{1:m}$. Because the complete order is DAG-consistent, each prefix is predecessor-closed. Its progress is $U_i(r_{1:m}) = W_i^{-1} \sum_{j=1}^{m} w_{iv_j}$, and its route length is reconstructed from the fixed start through the validated endpoint of each bound instance, using the same segment and doorway-routing policy as the complete references. For observed progress $q_i > 0$, $\mathcal{R}_i^{\mathrm{part}}(q_i)$ contains all such prefixes satisfying $U_i(r_{1:m}) = q_i$, and $L_i^*(q_i)$ is their minimum route length. The solution-conditioned partial reference instead follows the agent's credited requirements, realized bindings, and first legal completion order.

When an admissible credited trace is not already represented by a stored prefix, its route is reconstructed on demand under the same fixed-endpoint contract. If either an exact-progress candidate or a required route segment still cannot be constructed, the evaluator does not substitute a nearest-progress reference: PSPL and the corresponding solution-conditioned partial metrics are undefined for that episode, and an aggregate is reported only when the required episode-level values are available. Zero progress is assigned PSPL zero without a reference lookup, while $q_i = 1$ reduces to the complete reference. The scope of these minima is therefore the materialized candidate, endpoint, and routing contracts rather than an exhaustive search over all physically feasible trajectories.

The episode-level Success weighted by Path Length (SPL) and Partial Success weighted by Path Length (PSPL) are

$$\begin{aligned} \mathrm{SPL}_i &= S_i \frac{L_i^*}{\max(P_i, L_i^*)}, \\ \mathrm{PSPL}_i &= \begin{cases} q_i \dfrac{L_i^*(q_i)}{\max(P_i, L_i^*(q_i))}, & q_i > 0, \\ 0, & q_i = 0. \end{cases} \end{aligned} \tag{6}$$

When both the executed and reference translation lengths are zero, the path efficiency factor is defined as one. SPL jointly measures complete demand satisfaction and embodied execution efficiency by comparing the trajectory with the shortest validated complete realization across admissible bindings and legal orders. Because its success factor is zero for every incomplete episode, SPL cannot express whether a failed trajectory nevertheless made efficient, valid progress. PSPL addresses this limitation by applying the same path-efficiency principle to dependency-consistent partial progress and its predecessor-closed partial reference.

Together, the five metrics separate complementary aspects of DEMANDNAV. SR measures complete satisfaction, RCR measures valid functional coverage, and PSR measures dependency-consistent progress. SPL measures efficient complete realization, whereas PSPL measures efficient partial realization. This extends the standard SR/SPL view for single-target navigation to multi-requirement human demands without collapsing full completion, dependency compliance, and navigation efficiency into one score. All five metrics are macro averages over the same $N$ episodes.

### C.2 SUPPLEMENTARY METRICS

Oracle Success (OS) removes only the termination requirement. With $O_i = 1$ when the trajectory reaches a complete legal fulfillment state,

$$\mathrm{OS} = \frac{1}{N}\sum_i O_i, \qquad \mathrm{TerminationGap} = \mathrm{OS} - \mathrm{SR}. \tag{7}$$

The Order Gap is $\mathrm{RCR} - \mathrm{PSR}$. Both gaps are nonnegative under consistent definitions and aggregation, and may be zero. Rounding both terms to the same precision preserves their ordering. OS is not assumed to equal SR: equality means that no legally complete episode loses success solely because of termination, not that premature stopping never occurs.

Solution-conditioned metrics separate route execution from the selected realization. Let $L_i^{\mathrm{SC}}$ be the shortest reference length among complete realizations actually completed by trajectory $i$, and let $L_i^{\mathrm{PSC}}$ fix the credited partial functional-requirement set, bindings, and first legal completion order. Then

$$\begin{aligned} \text{SC-SPL}_i &= S_i \frac{L_i^{\mathrm{SC}}}{\max(P_i, L_i^{\mathrm{SC}})}, \qquad \mathrm{PE}_i = \frac{L_i^*}{L_i^{\mathrm{SC}}} \quad (S_i = 1), \\ \text{SC-PSPL}_i &= q_i \frac{L_i^{\mathrm{PSC}}}{\max(P_i, L_i^{\mathrm{PSC}})}, \quad \mathrm{PPE}_i = \frac{L_i^*(q_i)}{L_i^{\mathrm{PSC}}} \quad (q_i > 0). \end{aligned} \tag{8}$$

SC-SPL and SC-PSPL are macro-averaged over all episodes, with zero contribution outside their success or positive-progress condition. Planning Efficiency (PE) averages only successful episodes, and Partial Planning Efficiency (PPE) averages only episodes with $q_i > 0$. A ratio with two zero reference lengths is defined as one. These metrics do not give an unconditional multiplicative decomposition. For an eligible successful episode with positive reference lengths,

$$\text{SC-SPL}_i \mathrm{PE}_i = \frac{L_i^*}{\max(P_i, L_i^{\mathrm{SC}})}. \tag{9}$$

This equals $\mathrm{SPL}_i$ when the denominators agree. A sufficient condition is $P_i \geq L_i^{\mathrm{SC}} \geq L_i^*$. The analogous partial identity requires equal denominators, for example $P_i \geq L_i^{\mathrm{PSC}} \geq L_i^*(q_i)$. Fixed reference endpoints do not guarantee these conditions. Even where an episode-level identity holds, the product of separately averaged factors generally differs from the average of their products, including on the same eligible set. PE and PPE must therefore be computed per episode, not as ratios of aggregate scores. If no episode satisfies the conditioning event, the corresponding conditional mean is undefined rather than zero.

## D EXPERIMENTAL DETAILS AND ADDITIONAL RESULTS

### D.1 RUNTIME AND OBSERVATION SETTINGS

DEMANDAGENT is executed through the Codex runtime with a 272K-token context window. At each observation step, the environment renders four RGB views at a resolution of $1600 \times 900$, providing a panoramic observation around the agent. Each view has a $120^\circ$ horizontal field of view and is captured at yaw offsets of $0^\circ$, $90^\circ$, $180^\circ$, and $270^\circ$, corresponding to FRONT, RIGHT, BACK, and LEFT, respectively. These four views are returned together as the current visual observation used by the agent for scene understanding, target grounding, and subsequent navigation decisions.

To inspect targets outside the default viewing horizon, the agent can additionally invoke the LOOK_DOWN and LOOK_UP tools. LOOK_DOWN tilts the camera downward by $30^\circ$, whereas LOOK_UP tilts it upward by $30^\circ$. Each viewpoint-adjustment action produces a fresh four-view observation under the updated camera pitch, allowing the agent to inspect objects or free-space regions that are difficult to observe from the default viewpoint.

### D.2 Baseline Adaptation and Execution Settings

To evaluate existing navigation methods on DEMAND-BENCH, we replace their simulator interfaces with AI2-THOR compatible observation, pose, and action transports while retaining the released method-specific perception, mapping, planning, and control components as closely as possible. Where required, we also adapt external model-provider calls and add benchmark-side demand translation and success evaluation; we do not replace these methods with the memory, ledger, skill, or point-navigation interfaces used by DEMANDAGENT.

Here, *object goal* is the baselines' native term for a target object category, and *subgoal* denotes one stage of baseline navigation; neither denotes a DEMANDNAV functional requirement. Under the **Human Demand setting**, OpenFMNav and InstructNav retain their demand-conditioned interfaces: a GPT-5.5 adapter decomposes each instruction into a static ordered list of natural-language sub-demands, and OpenFMNav then uses its released proposal module to infer candidate object targets. SG-Nav, Uni-NaVid, ApexNav, WMNav, and CogNav instead use a shared GPT-5.5 object-goal adapter that converts the instruction into a static ordered sequence of object categories before navigation. At the simulator boundary, all adapted wrappers receive $640 \times 480$ egocentric observations, matching the released sensor configuration for most baselines, while retaining method-specific sensing modalities, preprocessing, and internal input resolutions; Uni-NaVid consumes RGB observations, whereas the other baselines consume RGB-D. Motion primitives and subgoal budgets likewise remain baseline-specific. Most discrete-action baselines use 0.25 m forward motions and $30^\circ$ turns as their primary locomotion primitives; SG-Nav additionally retains a $60^\circ$ turn, whereas WMNav retains its native polar actions, executed through exact-angle turns and movement chunks of at most 0.1 m. InstructNav uses 495 steps per sub-demand, while the other baselines use 500 steps per subgoal; for WMNav, this limit counts high-level policy decisions rather than individual simulator actions.

### D.3 Supplementary metric analysis

Table 8 complements the primary results in Table 1 with the five supplementary metrics defined in Appendix C.

Among the reported values, DEMANDAGENT has the highest OS, SC-SPL, and SC-PSPL under both input settings, and the highest Human Demand PPE. As conditional means, PE and PPE should be read jointly with SR and PSR. The equality OS $=$ SR rules out legally complete episodes losing success solely through termination in the reported aggregate, but it does not rule out premature stopping in unsuccessful episodes.

### D.4 Ablation settings and Token Cost

**Variant definitions.** **Vanilla Codex** retains the common Codex runtime, visual input, point-based navigation executor, budgets, completion-event adapter, and evaluator, but uses none of the ADL, Spatial Memory, or domain-specific skill components. The **w/o ADL** setting removes the explicit demand ledger, its operations, and the associated REPLAN capability. It retains the functional-requirement reasoning guidance and all other system components, including Spatial Memory and the remaining skills. Its observation schedule and evaluation settings are unchanged. **ADL w/o Replan** disables REPLAN while retaining ledger initialization and state updates. The **w/o Spatial Memory** condition removes explicit memory storage and recall. **w/o Skills** removes domain-specific procedural guidance. Tables 9 and 10 summarize the runtime, component, and evaluation settings.

The no-ADL variants follow the same demand-only initialization phase and receive the initial RGB observation at the same point in the episode as the ADL-enabled variants. They emit an equivalent explicit completion event when a target-directed action passes the completion checks, whereas ADL-enabled variants submit the event through a ledger update. All events enter the same evaluator and are scored under the same acceptance and success rules. Thus, w/o ADL removes explicit ledger state and ledger-dependent operations, but does not replace the task guidance or relax the scoring protocol.

**Performance by dependency structure.** We partition the 200 episodes using the reference annotation: single-requirement episodes contain one functional requirement, multi-requirement episodes without dependencies contain multiple requirements and no prerequisite edges, and dependency-

Table 8: **Supplementary results on DEMAND-BENCH.** Results are reported in %. PE and PPE are conditional means. The top-3 results are highlighted as first , second , and third .

| Method | Human Demand | | | | | Oracle Object Sequence | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | OS ↑ | SC-SPL ↑ | PE ↑ | SC-PSPL ↑ | PPE ↑ | OS ↑ | SC-SPL ↑ | PE ↑ | SC-PSPL ↑ | PPE ↑ |
| OpenFMNav (Kuang et al., 2024) | 6.5 | 4.7 | 94.5 | 11.0 | 78.0 | 10.0 | 7.5 | 95.0 | 13.5 | 80.0 |
| InstructNav (Long et al., 2024) | 7.5 | 5.3 | 96.3 | 12.6 | 67.4 | 11.0 | 8.7 | 97.2 | 14.4 | 72.7 |
| SG-Nav (Yin et al., 2024) | 6.5 | 4.3 | 95.5 | 10.2 | 73.1 | 9.0 | 6.6 | 94.1 | 12.7 | 77.7 |
| Uni-NaVid (Zhang et al., 2024) | 4.5 | 2.0 | 80.0 | 8.6 | 73.8 | 7.0 | 4.6 | 92.7 | 10.5 | 86.0 |
| ApexNav (Zhang et al., 2025) | 12.0 | 8.6 | 86.0 | 15.8 | 67.6 | 34.5 | 26.2 | 80.6 | 32.7 | 68.0 |
| WMNav (Nie et al., 2025) | 8.0 | 5.6 | 94.0 | 13.7 | 68.6 | 12.0 | 9.5 | 95.4 | 16.2 | 75.4 |
| CogNav (Cao et al., 2025) | 11.0 | 7.8 | 86.9 | 15.0 | 60.9 | 31.5 | 22.5 | 76.4 | 30.3 | 61.0 |
| **DEMANDAGENT (Ours)** | 52.5 | 29.4 | 86.9 | 32.4 | 85.7 | 76.0 | 49.4 | 83.2 | 51.7 | 81.9 |

Table 9: **Runtime and interface settings for the DEMAND-BENCH ablations.** All variants use GPT-5.6-Luna with max reasoning effort. Budget entries denote MCP tool calls/navigation decisions.

| Variant | Backbone / effort | Observation | Nav. executor | Budget |
|---|---|---|---|---|
| Vanilla Codex | GPT-5.6-Luna / max | Four-view RGB | Point-based | 500 / 200 |
| w/o ADL | GPT-5.6-Luna / max | Four-view RGB | Point-based | 500 / 200 |
| ADL w/o Replan | GPT-5.6-Luna / max | Four-view RGB | Point-based | 500 / 200 |
| w/o Spatial Memory | GPT-5.6-Luna / max | Four-view RGB | Point-based | 500 / 200 |
| w/o Skills | GPT-5.6-Luna / max | Four-view RGB | Point-based | 500 / 200 |
| Box-based Nav. | GPT-5.6-Luna / max | Four-view RGB + annotated view | Box-based | 500 / 200 |
| **DEMANDAGENT (Ours)** | GPT-5.6-Luna / max | Four-view RGB | Point-based | 500 / 200 |

bearing episodes contain multiple requirements and at least one prerequisite edge. Table 11 defines the purpose of each group and reports the corresponding results.

For single-requirement episodes, DEMANDAGENT succeeds on 63 of 70 episodes, so its SR, RCR, and PSR are all 90.0%. Performance decreases as the number of functional requirements and dependency constraints increase. On dependency-bearing episodes, both ADL-enabled variants substantially outperform the no-ADL variants. ADL w/o Replan performs better than the full system on this subset, while the full system achieves slightly higher aggregate performance over the complete benchmark.

The main point interface takes a normalized image point and delegates endpoint construction to the geometric executor. Box-based Nav. instead invokes a language-conditioned proposal stage, renders an annotated frozen view, and requires `select_nav_candidate` before execution. It is therefore a different grounding and interaction pipeline rather than the same point interface with a cosmetic representation change.

**Box-selection protocol and qualitative comparison.** For Box-based Nav. using LocateAnything (Wang et al., 2026b), the agent first emits a referring expression for a visible anchor. LocateAnything applies that expression to one frozen RGB view and returns one or more candidate boxes, which are overlaid with stable letter IDs. No motion occurs during proposal generation. The annotated view is then returned to the agent, which must inspect the pixels and call `select_nav_candidate` with exactly one candidate ID (or `NONE`). Only after this second-stage choice does the geometric executor construct and execute a collision-checked endpoint.

In the panel labels below, *goal* denotes a low-level navigation target, not a functional requirement. Figure 6 shows the resulting interface difference. In the multi-candidate example, the agent queried LocateAnything with "the framed desert landscape mounted on the gray wall." LocateAnything proposed two boxes: candidate A covers a mountain landscape, whereas candidate B covers the intended desert scene with airships. The agent therefore selected B after inspecting the labeled image. The point marker on the same observation illustrates the corresponding point-based input. In the lower example, the query "The hall" returned no candidate boxes, while the floor point shows how the point interface can directly specify a free-space region.

Table 10: **Component and execution settings for the DEMAND-BENCH ablations.** "Shared" denotes the same private requirement–object-instance matching and DAG-consistent scoring rule.

| Variant | ADL | REPLAN | Memory | Skills | Initialization / RGB release | Completion event / scoring |
|---|---|---|---|---|---|---|
| Vanilla Codex | Off | Off | Off | Off | Demand-only reasoning → RGB | Target-action event / Shared |
| w/o ADL | Off | Off | On | On | Demand-only reasoning → RGB | Target-action event / Shared |
| ADL w/o Replan | On | Off | On | On | Demand → ADL init → RGB | ADL update / Shared |
| w/o Spatial Memory | On | On | Off | On | Demand → ADL init → RGB | ADL update / Shared |
| w/o Skills | On | On | On | Off | Demand → ADL init → RGB | ADL update / Shared |
| Box-based Nav. | On | On | On | On | Demand → ADL init → RGB | ADL update / Shared |
| **DEMANDAGENT (Ours)** | On | On | On | On | Demand → ADL init → RGB | ADL update / Shared |

Table 11: **Performance by requirement and dependency structure on DEMAND-BENCH.** Results are reported in %. Episodes are partitioned into single-requirement tasks, multi-requirement tasks without prerequisite relations, and multi-requirement tasks with prerequisite relations. For groups without prerequisite relations, RCR and PSR are identical by definition. The top-3 results in each group and metric are highlighted as first , second , and third ; ties share the same rank.

| | **Single Req.** ($n = 70$) | | | **Multi. w/o Prereq.** ($n = 22$) | | | **Multi. w/ Prereq.** ($n = 108$) | | |
|---|---|---|---|---|---|---|---|---|---|
| **Method** | **SR** | **RCR** | **PSR** | **SR** | **RCR** | **PSR** | **SR** | **RCR** | **PSR** |
| Vanilla Codex | 78.6 | 78.6 | 78.6 | 22.7 | 50.0 | 50.0 | 2.8 | 8.4 | 8.4 |
| w/o ADL | 88.6 | 90.0 | 90.0 | 31.8 | 65.2 | 65.2 | 13.9 | 29.5 | 29.5 |
| ADL w/o Replan | 68.6 | 81.4 | 81.4 | 59.1 | 68.0 | 68.0 | 38.9 | 44.9 | 44.9 |
| **DEMANDAGENT (Ours)** | 90.0 | 90.0 | 90.0 | 54.5 | 79.5 | 79.5 | 27.8 | 38.6 | 37.9 |

This comparison helps explain the system-level ablation in Table 2. Box-based Nav. attains a slightly higher RCR than the point-based system (63.0% versus 61.1%), yet substantially lower SR (39.0% versus 52.5%), PSR (54.2% versus 60.7%), SPL (16.5% versus 26.1%), and PSPL (21.3% versus 28.4%). Higher RCR does not by itself establish that more objects were perceived. The point interface accepts objects, doorway openings, or free space without a model-facing proposal-selection round, but remains susceptible to visual grounding errors. The comparison concerns the complete grounding-and-interaction pipeline and does not isolate point versus box representation or the proposal model alone.

**Input-token accounting.** Figure 7 compares success rate with mean cumulative input-token usage per episode. These counters do not measure monetary cost or include external proposal-model requests.

### D.5 EB-NAVIGATION ADAPTATION AND COMPLETE RESULTS

**Benchmark protocol.** Our local evaluation uses 60 episodes in each of EB-Navigation's five subsets. Its native interface supplies one $500 \times 500$ front RGB observation after each of eight discrete actions and imposes a 20-step budget. Success requires a horizontal distance of at most 1 m from the private target position. The policy receives no target coordinate or distance, depth, segmentation, map, or reachable-point oracle. The adapted condition uses Codex CLI 0.153.4 with GPT-5.6-Luna at maximum reasoning effort, an EB-specific object-search skill, ADL, and Spatial Memory. It retains EB-Navigation's native discrete-action interface rather than the point-navigation executor. The direct condition uses the native EB-Navigation evaluation interface. This is an adapted system-level comparison rather than an isolated component ablation.

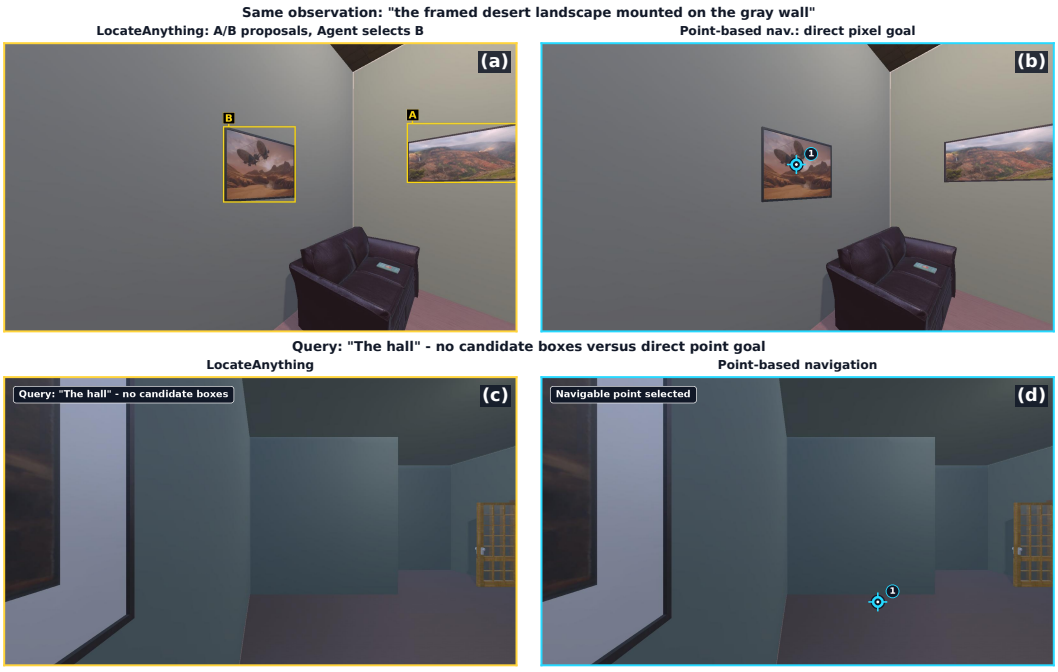


Figure 6: **Qualitative comparison of box-based and point-based navigation interfaces.** (a) For the query "the framed desert landscape mounted on the gray wall," LocateAnything returns two labeled proposals and the agent selects candidate B. (b) On the identical frozen observation, an illustrative point marks the center of candidate B. (c) For the agent query "The hall," LocateAnything returns no candidate boxes, so no candidate ID can be selected. (d) An illustrative floor point shows the alternative input representation.

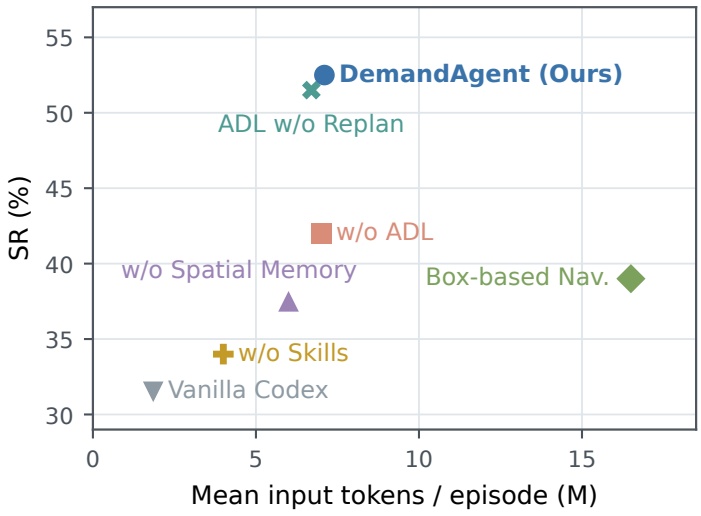


Figure 7: **Success rate versus mean input tokens per episode.** SR and cumulative input tokens are reported in % and millions (M), respectively.

The largest gain over the direct GPT-5.6-Luna condition occurs on Long Horizon. Because the two conditions differ at the system level, this comparison does not isolate the contribution of reasoning, memory, or perception.

Table 12: **Complete EB-Navigation success rates (%).** Published baseline values are from Yang et al. (2025), while the final two rows show the direct and adapted GPT-5.6-Luna conditions. Bold and underlined entries mark the best and second-best results, respectively, with ties marked jointly.

| Model | EB-Navigation | | | | | |
|---|---|---|---|---|---|---|
| | **Avg** | **Base** | **Common** | **Complex** | **Visual** | **Long** |
| *Proprietary MLLMs* | | | | | | |
| GPT-4o | 57.7 | 55.0 | 60.0 | 58.3 | **60.0** | <u>55.0</u> |
| GPT-4o-mini | 32.8 | 31.7 | 33.3 | 35.0 | 28.3 | 33.3 |
| Claude-3.7-Sonnet | 45.0 | 50.0 | 61.7 | 50.0 | 36.7 | 26.7 |
| Claude-3.5-Sonnet | 44.7 | 66.7 | 51.7 | 41.7 | 36.7 | 26.7 |
| Gemini-1.5-Pro | 24.3 | 23.3 | 25.0 | 25.0 | 28.3 | 20.0 |
| Gemini-2.0-flash | 48.7 | 63.3 | <u>65.0</u> | 50.0 | 51.7 | 13.3 |
| Gemini-1.5-Flash | 41.7 | 56.7 | 50.0 | 46.7 | 50.0 | 5.0 |
| Qwen-VL-Max | 39.7 | 50.0 | 46.7 | 41.7 | 35.0 | 25.0 |
| GPT-4o (Lang.) | 17.4 | 21.7 | 21.7 | 26.7 | 16.7 | 0.0 |
| GPT-4o-mini (Lang.) | 8.3 | 3.3 | 13.3 | 10.0 | 15.0 | 0.0 |
| *Open-Source MLLMs* | | | | | | |
| Llama-3.2-90B-Vision-Ins | 30.0 | 48.3 | 23.3 | 38.3 | 33.3 | 6.7 |
| Llama-3.2-11B-Vision-Ins | 21.4 | 23.3 | 21.7 | 26.7 | 18.3 | 17.0 |
| InternVL2.5-78B | 30.7 | 36.7 | 38.3 | 33.3 | 21.7 | 23.3 |
| InternVL2.5-38B | 30.3 | 35.0 | 28.3 | 38.3 | 26.7 | 23.3 |
| InternVL2.5-8B | 21.3 | 35.0 | 23.3 | 21.7 | 26.7 | 0.0 |
| InternVL3-78B | 53.7 | 66.7 | 63.3 | 61.7 | 45.0 | 31.7 |
| InternVL3-38B | 50.7 | 55.0 | 61.7 | 55.0 | <u>56.7</u> | 25.0 |
| InternVL3-8B | 29.3 | 38.3 | 30.0 | 40.0 | 33.3 | 5.0 |
| Qwen2-VL-72B-Ins | 21.2 | 26.7 | 30.0 | 28.3 | 16.0 | 5.0 |
| Qwen2-VL-7B-Ins | 14.0 | 26.7 | 10.0 | 15.0 | 15.0 | 3.3 |
| Qwen2.5-VL-72B-Ins | 40.0 | 46.7 | 46.7 | 46.7 | 26.7 | 33.3 |
| Qwen2.5-VL-7B-Ins | 20.3 | 20.0 | 26.7 | 38.3 | 16.7 | 0.0 |
| Ovis2-34B | 45.7 | 63.3 | 50.0 | 56.7 | 46.7 | 11.7 |
| Ovis2-16B | 47.7 | 60.0 | 46.7 | 58.3 | 48.3 | 25.0 |
| gemma-3-27b-it | 45.4 | 53.3 | 45.0 | 61.7 | 50.0 | 16.7 |
| gemma-3-12b-it | 34.0 | 38.3 | 36.7 | 48.3 | 40.0 | 6.7 |
| *Our evaluated systems* | | | | | | |
| **GPT-5.6-Luna** | <u>59.3</u> | <u>68.3</u> | **66.7** | <u>68.3</u> | **60.0** | 33.3 |
| **DemandAgent (adapted)** | **65.7** | **71.7** | **66.7** | **70.0** | **60.0** | **60.0** |