SMH-BENCH: BENCHMARKING LLM AGENTS FOR ENVIRONMENT-GROUNDED REASONING AND ACTION IN SMART HOMES

Anonymous authors
Paper under double-blind review

ABSTRACT

Smart homes are evolving toward complex state-dependent living environments, requiring Large Language Models (LLMs) to reason over user intent, preferences, and multi-device interactions. However, existing smart-home benchmarks often focus on static instruction-to-API mapping or limited simulations, failing to evaluate whether LLMs can reason, interact, and act reliably in realistic household scenarios. To address these limitations, we introduce SMH-Bench, a comprehensive benchmark for evaluating LLMs in smart-home environments. Built upon HomeEnv, an executable and verifiable smart-home simulator, SMH-Bench contains 1,100 high-quality tasks spanning 7 categories and 22 fine-grained subcategories. It further stratifies tasks across simple, medium and complex homes, ranging from small apartments to dense multi-room environments with 135 devices. Experiments show that although frontier LLMs achieve strong performance on explicit control and query tasks, they still exhibit significant weaknesses in automation task scheduling, ambiguity handling and personalized reasoning, especially as home complexity increases. We hope SMH-Bench will facilitate the development of more reliable, context-aware, and practically deployable smart-home agents.

1 INTRODUCTION

Smart homes are evolving toward increasingly complex and personalized user demands, requiring systems to understand natural language instructions, infer implicit user intent, and coordinate actions across multiple devices and rooms. Recently, large language model (LLM)-driven agents have demonstrated significant potential in intent understanding (Wang et al., 2025; Zhang et al., 2025), tool invocation (Liu et al., 2025; Patil et al., 2025), and multi-step decision-making (Shridhar et al., 2021; Yao et al., 2022; Kim et al., 2025), making them a promising foundation for complex smart home tasks (King et al., 2024; Rivkin et al., 2025; Yin et al., 2025). Consequently, rigorously evaluating these agents in authentic, complex smart home settings has emerged as a critical challenge.

While current evaluation benchmarks have fueled progress in this field, they still exhibit notable limitations. Many benchmarks (King et al., 2024; Li et al., 2025) formulate tasks as static instruction-to-API mappings, which evaluates basic comprehension but ignores dynamic home states and fails to verify success through environmental feedback. Recent works (Rivkin et al., 2025; Seo et al., 2026) further introduce state-aware simulations to ground evaluations in interactive environments. However, their tasks remain confined to limited scenarios like simple queries and device control, leaving the complex, continuous, and personalized demands of real-world interactions underexplored. In practice, user instructions are rarely isolated, fully specified single-turn commands. Instead, they frequently span multiple devices and rooms, requiring agents to disambiguate intent based on real-time states, dialogue context, and user preferences. Moreover, as home ecosystems scale, agents must perform complex querying, filtering, and compositional control across extensive device networks. Ultimately, existing benchmarks still fall short of comprehensively evaluating whether agents can understand, reason, remember, and act in realistic homes.

To address these limitations, we introduce SMH-Bench, a more comprehensive benchmark for evaluating smart-home agents in realistic household settings. With the goal of defining, executing, and

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.

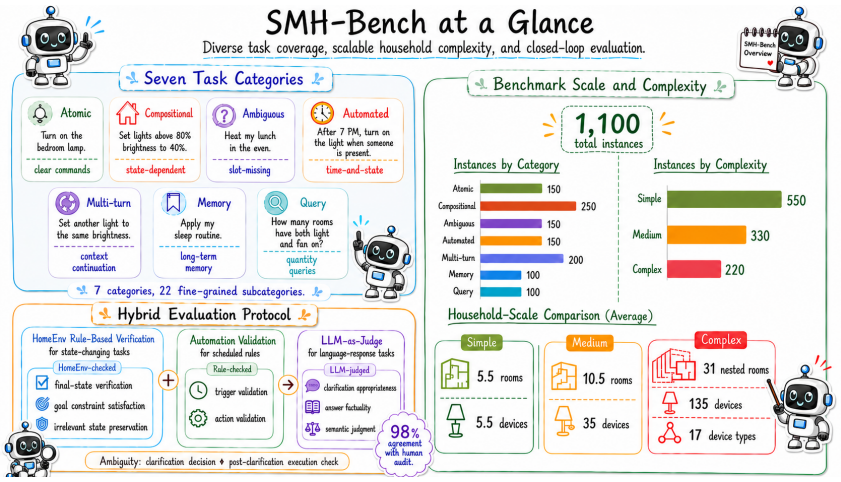


Figure 1: Overview of SMH-Bench. SMH-Bench evaluates agents with 1,100 human-audited tasks across seven capability categories, scalable household complexity, and a hybrid evaluation protocol.

Table 1: Comparing SMH-Bench to existing benchmarks for smart-home evaluation.

Benchmark	Device Query	Automation	Clarify Necessity	Spoken / Correction	Multi-turn Dialogue	Personalized Memory	Dataset Size
HomeBench	✗	✗	✗	✗	✗	✗	170K
SimuHome	✓	✓	✓	✗	✗	✗	600
SMH-Bench	✓	✓	✓	✓	✓	✓	1,100

verifying diverse task types within a unified framework, we design HomeEnv, a stateful, executable, and verifiable smart home environment, as its foundational infrastructure. HomeEnv models the home as a continuously evolving state, encompassing rooms, devices, and their dynamic attributes. Unlike static API-based evaluations, HomeEnv allows agents to interact with executable home states and verifies task success based on the actual state transitions caused by agent actions. This design enables flexible evaluation of heterogeneous tasks without relying on rigid reference action sequences. Table 1 compares SMH-Bench with existing smart-home benchmarks.

Building upon HomeEnv, we introduce a task hierarchy designed to evaluate the core reasoning and execution capabilities of smart-home agents. SMH-Bench covers diverse dimensions including linguistic robustness, state-driven reasoning, multi-turn interaction, and personalized memory. For example, a user might remark that "it feels a bit chilly, so please adjust the heating in the rooms I usually use to the same temperature as last night." Fulfilling this requires the agent to query real-time environmental variables, retrieve preferences from historical dialogue, and identify the user's frequently occupied rooms. In total, we construct 1,100 tasks across 7 categories, spanning household scales from simple apartments with an average of 5.5 devices to complex environments with 135 devices, as detailed in Figure 1.

We comprehensively evaluate 13 representative LLMs using SMH-Bench, revealing a persistent capability gap in authentic smart home interactions. Although Gemini-3.1-Pro achieves an overall success rate of 85.2% under EIA, performance remains highly inconsistent across task complexities. While agents reliably handle explicit atomic controls, they experience severe performance degradation in complex scenarios requiring ambiguous intent resolution, compositional multi-device coordination, state-driven automation, and personalized memory utilization.

2 Related Work

LLM Agents in Smart Homes. LLM-based smart-home agents have broadened smart-home interaction from command parsing to context-grounded decision making. Earlier studies on smart speak-

ers and home assistants examined long-term use, proactive interaction, and privacy risks (Mckie et al., 2022; Zargham et al., 2022; Lutz & Newlands, 2021), while recent LLM-based systems study several concrete capabilities. Sasha (King et al., 2024) and SAGE (Rivkin et al., 2023) focus on ambiguous goal reasoning and interaction with household context and device APIs. ChatIoT (Li et al., 2023) and AwareAuto (Shi et al., 2024) translate natural-language instructions into automation rules, and IoTGPT (Yu et al., 2026), HomeLLaMA (Huang et al., 2025), and Harmony (Yin et al., 2025) further consider memory, on-device execution, latency, and privacy. These studies show that LLM-based smart-home agents are increasingly defined not by a single function such as command parsing or device control, but by the coordination of multiple capabilities, including intent grounding, device/API interaction, automation, memory, personalization, and reasoning over household-specific constraints.

Evaluation and Benchmarks in Smart Homes. Smart-home benchmarks have evolved from activity recognition and embodied household tasks toward LLM-based assistant evaluation. CASAS (Cook et al., 2013) and ARAS (Alemdar et al., 2013) focus on activity recognition, while VirtualHome (Puig et al., 2018), ALFRED (Shridhar et al., 2020), TEACh (Padmakumar et al., 2022), BEHAVIOR (Srivastava et al., 2022), and ReALFRED (Kim et al., 2024) emphasize navigation and physical manipulation in household environments. Recent smart-home benchmarks are closer to our setting: HomeBench (Li et al., 2025) evaluates valid and invalid device-control instructions, SmartHome-Bench (Zhao et al., 2025) and SmartBench (Zou et al., 2026) study anomaly and safety-related scenarios, and SimuHome (Seo et al., 2026) introduces executable temporal simulation. Compared with these efforts, SMH-Bench expands the evaluation scope from isolated device control or safety scenarios to a broader set of environment-grounded tasks involving compositional control, ambiguity resolution, automation, dialogue context, memory, and state queries, while using HomeEnv to execute agent actions and verify final-state correctness, preservation of irrelevant states, and task-specific outcomes such as automation conditions, clarification decisions, and query answers across homes of different complexity.

3 SMH-BENCH

SMH-Bench aims to provide a comprehensive and realistic evaluation benchmark for LLM-driven agents in smart-home environments. To ground agent evaluation in concrete household contexts, we build a simulated smart-home environment, named HomeEnv, which models room layouts, device configurations, and dynamic device states. In this section, we first formalize the interaction task for smart-home agents and then describe the design of HomeEnv. We further present the benchmark taxonomy, data construction pipeline, dataset statistics, and evaluation protocol. Figure 2 illustrates the construction process of SMH-Bench.

3.1 TASK FORMULATION

We formulate smart-home agent evaluation as an environment-grounded interaction task. At interaction turn t, the smart home is represented as $\mathcal{H}_t = (\mathcal{R}, \mathcal{D}, \phi, \mathcal{X}_t, \mathcal{S})$, where $\mathcal{R}$ is the set of rooms, $\mathcal{D}$ is the set of devices, $\phi(d_j) \in \mathcal{R}$ maps each device d_j to its located room, $\mathcal{X}_t(d_j)$ denotes the dynamic attributes of d_j, and $\mathcal{S}(d_j)$ denotes its callable services with corresponding arguments. Given a user instruction u, the agent needs to ground the instruction in the current home state and decide the appropriate response.

For executable control tasks, the agent produces one or more service calls. A single action is denoted as $a = [d_j.s_k(\mathbf{p})]$, where d_j is the target device, $s_k \in \mathcal{S}(d_j)$ is a callable service, and $\mathbf{p}$ denotes the service arguments. Executing an action updates the environment through $\mathcal{H}_{t+1} = T(\mathcal{H}_t, a)$. For tasks involving multiple operations, the agent outputs an action sequence $A = (a_1, \ldots, a_n)$, which induces a sequence of state transitions from the initial state $\mathcal{H}_0$ to the final state $\mathcal{H}_n$.

To cover realistic household interactions, each benchmark instance is defined as $\tau = (\mathcal{H}_0, u, \mathcal{C}, \mathcal{M}, g)$, where $\mathcal{C}$ is the dialogue history, $\mathcal{M}$ is user memory, and g is the task-specific goal. Depending on the task type, g may specify target device states, expected query answers, clarification requirements, or scheduled execution conditions. Accordingly, the agent output can be executable service calls, a natural-language answer, a clarification question, or a scheduled rule. A task is considered successful if the agent output satisfies g under the current environment: executable tasks

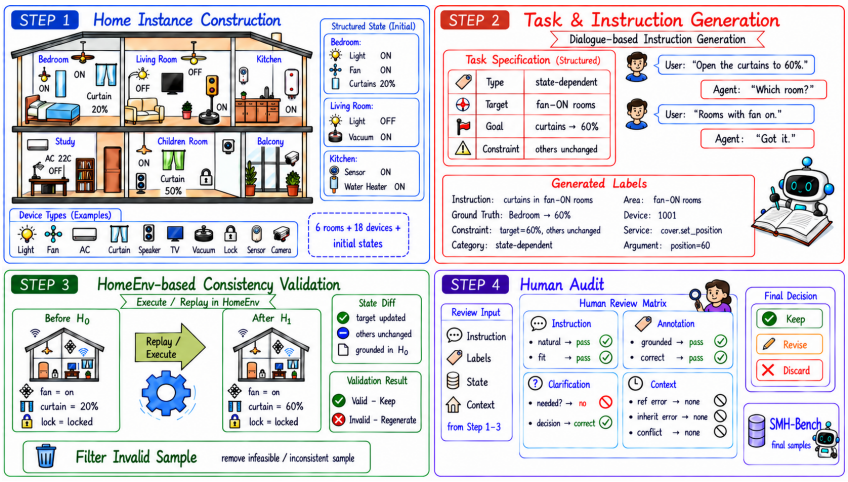


Figure 2: SMH-Bench construction pipeline. Each instance is built through four stages: home instance construction, task and instruction generation from structured home states, HomeEnv-based consistency validation, and final human audit before inclusion in the benchmark.

are verified by whether the final state $\mathcal{H}_n$ satisfies the target constraints while preserving irrelevant device states, whereas non-executable responses are evaluated by task-specific criteria described in Section 3.4.

3.2 HomeEnv: Simulated Smart-Home Environment

HomeEnv is a lightweight simulated smart-home environment designed to support executable agent interactions in realistic household settings. It serves as the environment foundation of SMH-Bench by providing structured home states, configurable device abstractions, and standardized query and control interface

Smart-Home State Space. HomeEnv represents a home as a structured environment composed of rooms and devices. Rooms define the spatial organization of the household and may contain nested structures, such as a bathroom inside a bedroom. Devices are assigned to rooms and described by their type and location. Each device is further represented through attributes and services: attributes describe current device states, such as power status, brightness, target temperature, humidity, or operation mode, while services define executable operations, such as turning a device on or off, setting a temperature, adjusting brightness, or switching modes. The full schema of rooms, devices, attributes, and services is provided in Appendix A.

Device Operation Engine. The operation engine updates the home state by executing structured actions issued by the agent. For each device-control action, the agent specifies the target device, the intended operation, and its parameters. HomeEnv checks whether the requested operation is valid for the target device, including whether the operation exists and whether the provided parameters satisfy predefined type, range, and option constraints. If the action is valid, the engine applies the operation and updates the corresponding device attributes; otherwise, the invalid operation is rejected and the home state remains unchanged.

Agent-Environment Interface. Agents interact with HomeEnv through structured query and control actions. The query interface allows agents to inspect the simulated home before acting, such as retrieving the room layout, finding devices in a specific room, listing devices of a certain type, checking their current states, or obtaining the operations that can be applied to them. The control interface allows agents to execute device operations through structured action objects. After each interaction, HomeEnv returns an observation containing the requested query result or the execution status of the control action. This standardized interface enables agents to perform perception and action within the same simulated home state.

3.3 BENCHMARK CONSTRUCTION

3.3.1 TASK TAXONOMY

To evaluate smart-home agents beyond device-type-specific command execution, SMH-Bench organizes tasks by environment-grounded capabilities. The taxonomy contains seven primary categories and 22 fine-grained subcategories. Atomic Control (TC1) covers single-intent operations under clear, colloquial, or noisy wording, while Compositional Control (TC2) evaluates multi-device, state-dependent, room-dependent, and selection-based execution. For example, "*except for the hallway, turn on all fans and air conditioners in rooms whose humidity is above 60%*" requires the agent to first identify rooms satisfying a sensor predicate and then apply actions only to devices located in those rooms.

Beyond explicit execution, SMH-Bench evaluates whether agents can handle ambiguity, automation, dialogue context, memory, and state queries. Ambiguous Intent (TC3) covers subjective complaints, scene-level goals, and slot-missing requests; for instance, "*the dining room feels like a steamer*" requires mapping a discomfort complaint to a feasible cooling intervention or asking for clarification when the state is underspecified. Automated Task Scheduling (TC4) evaluates persistent time- or state-triggered rules, such as turning on hallway and staircase lights at a specified brightness when motion is detected in selected rooms. Context-Aware Multi-turn Interaction (TC5) tests whether the current turn inherits, revises, or replaces previous operations, while Personalized Memory (TC6) requires applying short- or long-term user preferences, such as setting another air conditioner "*the same way as before*". Environment-Grounded Query (TC7) evaluates non-mutating inspection and aggregation over the current home state. Together, these categories stress-test state grounding, target selection, clarification, rule construction, dialogue tracking, memory use, query reasoning, and irrelevant-state preservation. Full subcategory definitions, representative examples, and evaluation targets are provided in Appendix C.

3.3.2 DATA CONSTRUCTION PIPELINE

SMH-Bench is constructed with an environment-first pipeline, as shown in Figure 2. We first instantiate concrete home states and then derive tasks, instructions, contexts, and labels from these states. This design makes each instance directly verifiable against HomeEnv.

Step1: Home Instance Construction. We construct simple, medium, and complex homes with different room layouts and device configurations. Each home specifies device locations, initial states, and available services for subsequent task generation.

Step2: Task and Instruction Generation. For each home, GPT-5 first generates a natural user instruction from the home state, task templates, and constraints, incorporating dialogue history or user memory as needed. A second GPT-5 call uses this instruction, state, and context to generate reference operations, automation conditions, query answers, or clarification responses. Detailed generation rules and examples are provided in Appendix B.

Step3: HomeEnv-based Consistency Validation. We automatically validate generated instances with HomeEnv before human review. For executable tasks, HomeEnv replays the reference operations to extract state-change labels and checks device existence, service availability, argument validity, and whether the reference operations reach the intended final state. For non-executable tasks, it recomputes query answers, verifies scheduling triggers and actions, and checks whether ambiguous requests indeed require clarification under the current state. Instances with state-label inconsistencies are revised or removed.

Step4: Human Audit. Finally, a pool of five annotators reviews the remaining instances for instruction naturalness, category alignment, and context or memory conflicts. Each instance is independently reviewed by three annotators and is included only after receiving unanimous approval from all three. In the first review round, 23% of candidate instances did not receive unanimous approval; these instances were revised and re-reviewed or removed. This final audit targets semantic and pragmatic errors that are difficult to catch through simulator checks alone.

3.3.3 Data Statistics and Analysis

SMH-Bench contains 1,100 human-audited evaluation instances across the taxonomy defined in Section 3.3.1. As shown in Figure 3, the dataset is not uniformly distributed across categories. Compositional Control and Context-Aware Multi-turn Interaction form the two largest groups, with 250 and 200 instances, respectively. Atomic Control, Ambiguous Intent, and Automated Task Scheduling each contain 150 instances, while Personalized Memory and Environment-Grounded Query each contain 100 instances. This allocation reflects the benchmark design choice to retain broad category coverage while assigning more evaluation mass to interaction settings that involve richer environment dependence.

SMH-Bench further stratifies instances by environmental complexity into simple, medium, and complex settings, with 550, 330, and 220 instances, respectively. As illustrated in Figure 1, these settings differ in spatial scale and device density. Simple environments model small apartments with an average of 5.5 rooms and 5.5 devices, while medium environments model multi-room homes with an average of 10.5 rooms and 35 devices. The complex setting has 31 nested rooms and 135 devices across 17 device types. This stratification provides a controlled axis for analyzing how agent performance changes as the observable home state becomes larger and more distractor-rich.



Figure 3: Task distribution of SMH-Bench. The benchmark organizes 1,100 instances into seven primary categories and 22 fine-grained subcategories.

3.4 Evaluation Protocol

SMH-Bench adopts a hybrid evaluation protocol that combines rule-based verification with LLM-as-a-Judge assessment. We use rule-based evaluation for tasks that can be verified in HomeEnv, and LLM-as-a-Judge evaluation for tasks that require semantic assessment.

Rule-Based Verification. For tasks involving physical environment changes, the HomeEnv simulator objectively verifies outcomes by comparing the post-execution state against the ground-truth goal. This process enforces an *irrelevant state preservation constraint* to ensure that devices not explicitly targeted remain strictly unchanged. Notably, for automated task scheduling (TC4), we implement a dual-rule validation mechanism: (1) *Trigger Validation*, where temporal triggers are verified by matching Cron trigger times, and state-driven triggers are evaluated by ensuring they trigger under the exact same device states as the ground truth; and (2) *Action Validation*, which verifies the resulting operations using the aforementioned state comparison.

LLM-as-a-Judge Assessment. For tasks where the agent's output is a natural language response, such as ambiguous intent (TC3) and environment-grounded queries (TC7), we utilize GPT-5 to assess the appropriateness of clarification requests and the factual accuracy of the answers. Separately from the construction audit, we conducted a blind human evaluation on 200 randomly sampled instances, obtaining 98.0% agreement between human judgments and the GPT-5 judge. We also re-judged all TC3/TC7 outputs using DeepSeek-V4-Pro, an independent-family judge, obtaining 96.4% agreement with GPT-5. The full stratified analysis is provided in Appendix G.

4 Experiments

4.1 Experimental Setup

We evaluate 13 representative LLMs on SMH-Bench under two settings. The first is Direct Reasoning (DR), where the model receives a global summary of the home state and the user query, then directly outputs the final prediction, such as structured actions, automation triggers, or natural-language responses. Predicted actions are executed in HomeEnv for outcome verification. The

Table 2: Performance comparison under EIA and DR settings. TC1–TC7 are defined in Section 3.3.1. For each model, success rates on individual capabilities and the overall average are reported in percentages. Token usage is reported in thousands. Δ Avg. measures the average success-rate difference between EIA and DR, where positive values indicate that EIA improves over DR.

Model	Setting	Capability (Success Rate, %)								Tokens (k) (↓)	Δ Avg. (EIA–DR)
		TC1	TC2	TC3	TC4	TC5	TC6	TC7	Avg.		
Open-source Models											
Qwen3.5-4B	DR	68.7	35.6	44.7	24.0	48.0	39.0	75.0	45.9	1.2	+10.6
	EIA	69.3	45.6	72.0	21.3	65.0	57.0	76.0	56.5	8.3	
Qwen3.5-9B	DR	73.3	38.0	47.3	25.3	51.5	47.0	77.0	49.2	1.4	+15.1
	EIA	76.7	56.0	69.3	36.0	74.0	65.0	81.0	64.3	7.4	
Qwen3.5-397B	DR	92.7	72.0	76.7	60.0	85.0	75.0	84.0	77.5	6.0	+0.3
	EIA	88.0	68.8	86.7	69.3	74.5	77.0	92.0	77.8	7.3	
DeepSeek-V3.2	DR	79.3	60.8	65.3	52.0	80.0	68.0	89.0	69.5	1.4	+5.5
	EIA	87.3	52.8	85.3	64.0	85.0	74.0	94.0	75.0	12.9	
MiniMax-M2.7	DR	81.3	55.6	60.7	38.7	32.5	32.0	63.0	51.8	2.5	−7.7
	EIA	57.3	23.6	52.0	42.0	41.0	27.0	90.0	44.1	4.7	
GLM-5	DR	80.7	50.0	49.3	52.0	71.0	39.0	78.0	59.7	3.1	+10.8
	EIA	88.7	65.6	83.3	42.7	76.0	51.0	87.0	70.5	3.6	
Closed-source Models											
Qwen3.5-Plus	DR	79.3	51.6	51.3	50.0	74.0	71.0	69.0	62.5	5.2	+7.5
	EIA	91.3	57.2	76.7	42.7	79.5	65.0	87.0	70.0	8.3	
GPT-5.4-Mini	DR	80.7	66.0	61.3	45.3	79.0	62.0	89.0	68.6	4.9	+6.7
	EIA	89.3	68.8	86.0	50.7	81.0	69.0	86.0	75.3	3.3	
GPT-5.4	DR	91.3	78.8	78.0	65.3	85.5	75.0	95.0	80.9	1.8	−1.4
	EIA	94.0	78.0	84.7	49.3	86.5	74.0	91.0	79.5	3.5	
Gemini-3.1-Flash	DR	89.3	73.6	63.3	62.7	72.5	71.0	91.0	74.0	0.9	+1.6
	EIA	86.7	69.6	84.7	48.7	80.0	73.0	95.0	75.6	3.1	
Gemini-3.1-Pro	DR	95.3	81.2	72.7	70.7	87.5	88.0	94.0	83.5	6.0	+1.7
	EIA	94.0	84.4	92.0	64.0	88.5	86.0	88.0	85.2	3.4	
Claude-Haiku-4.5	DR	90.0	73.2	72.7	64.0	82.0	77.0	79.0	76.6	2.5	−0.4
	EIA	88.0	73.6	85.3	48.0	81.0	73.0	87.0	76.2	3.1	
Claude-Sonnet-4.6	DR	90.7	73.2	61.3	68.7	84.0	77.0	85.0	76.7	1.2	+7.4
	EIA	93.3	81.2	91.3	68.7	85.0	80.0	92.0	84.1	3.4	

second setting is the Environment-Interactive Agent (EIA), which follows the ReAct paradigm and interacts with HomeEnv through iterative observations and actions. Unlike Direct Reasoning, the agent starts with partial observability, such as basic room and device lists, and must use tools to query attributes, retrieve services, and execute operations. All models are evaluated under EIA with an interaction budget of 10 rounds. This setting better reflects real-world smart-home deployment. Implementation details for both settings are provided in Appendix D.1.

4.2 Main Results

Overall Performance Table 2 reports the main results of all evaluated models under the DR and EIA settings across seven capability dimensions. Gemini-3.1-Pro achieves an average success rate of 85.2% under EIA and 83.5% under DR. Among open-weight models, DeepSeek-V3.2 reaches 75.0% under EIA and 69.5% under DR, placing it on par with frontier compact variants. Five paired runs further support the consistency of DeepSeek-V3.2's overall EIA advantage (Appendix F.2).

Capability-wise Insights. Across capability dimensions, model differences are most visible once tasks move beyond direct device access. TC1 and TC7 represent the most approachable categories; for instance, Gemini-3.1-Pro achieves 94.0% and 88.0% under EIA respectively, and even our smallest evaluated model, Qwen3.5-4B, reaches a competitive 69.3% and 76.0%. However, the performance gap widens drastically on tasks requiring more complex reasoning. For example, DeepSeek-V3.2's EIA performance drops from 87.3% on TC1 to merely 52.8% on TC2. Furthermore, TC4 emerges as a common bottleneck across models. Even the top-performing Gemini-3.1-Pro only manages 64.0% under EIA and 70.7% under DR on TC4, while our smallest evaluated model Qwen3.5-4B plummets to 21.3% under EIA. These results highlight the multidimensional difficulty of SMH-

Bench, validating its hierarchical design, which rigorously separates atomic device operation from high-level spatial, temporal, and compositional reasoning.

Execution Mode Comparison. The EIA setting broadly benefits open-source and mid-tier models by enabling iterative exploration and real-time state querying. For instance, Qwen3.5-9B and GLM-5 exhibit massive average improvements of 15.1 and 10.8 percentage points under EIA, respectively. EIA is particularly critical for TC3, where models can iteratively clarify uncertainty; notably, Claude-Sonnet-4.6 jumps from 61.3% under DR to 91.3% under EIA in this category. Conversely, DR proves highly competitive for certain leading models with robust long-context integration. GPT-5.4 achieves overall success rates of 80.9% under DR and 79.5% under EIA, with TC4 scores of 65.3% under DR versus 49.3% under EIA. This contrast suggests that while EIA strengthens interactive grounding, it may introduce multi-step cascading errors, whereas DR relies entirely on the model's single-pass global reasoning.

4.3 Further Analysis

Impact of Home Complexity. LLMs become less reliable as home environments grow more complex. As shown in Figure 4, all representative models exhibit a clear performance drop from simple to complex homes in both settings. In the DR setting, Gemini-3.1-Pro decreases from 88.9% to 76.8%, while Qwen3.5-9B drops more sharply from 59.8% to 33.6%. A similar trend appears in the EIA setting. These results suggest that home complexity mainly hurts models by expanding the space of candidate rooms, devices, states, and constraints. Tool access helps retrieve states and locate devices, but it does not solve the higher-level problem of deciding what to query, what to ignore, and when to stop. The gap therefore shows that current agents still struggle to scale to realistic homes, where execution requires robust grounding, state filtering, and constraint satisfaction.

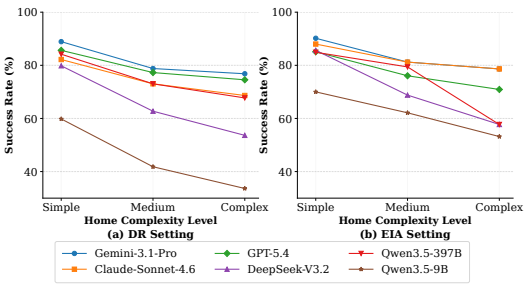


Figure 4: Pass rates of six models across home complexity levels in DR and EIA settings.

When EIA Compensates for Thinking. As shown in Figure 5, removing thinking hurts DeepSeek-V3.2 much more in DR than in EIA, with the overall drop increasing from 7.3 points in EIA to 18.3 points in DR. The non-thinking variant remains relatively competitive in EIA on atomic control and device query, suggesting that tool calls can externalize state retrieval and action execution. In contrast, the largest drops in DR appear on memory, query, composite, and multi-turn tasks, where the model must internally select relevant evidence, maintain user preferences, and integrate information across steps. This contrast shows that tool interaction can compensate for weak reasoning when tasks depend mainly on observable states and explicit actions, but thinking remains crucial for latent preferences, multi-step coordination, and deciding what information to query or ignore.

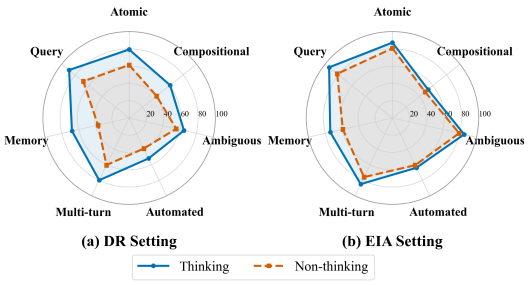
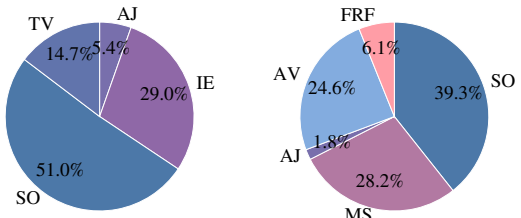


Figure 5: Comparison of DeepSeek-V3.2 Thinking and Non-thinking modes across task categories under the DR and EIA settings.

Category-level Execution Mode Differences. Table 19 shows that the relative performance of DR and EIA varies across task categories, models, and home-complexity levels. EIA generally achieves higher success rates on ambiguous-intent and environment-grounded query tasks. For atomic and compositional control, automation, multi-turn interaction, and personalized memory, the direction and magnitude of the gap depend on the model and complexity level. These results indicate that neither execution setting is uniformly better across all task conditions.

Table 3: Error taxonomy. Detailed descriptions and cases are provided in Appendix E.

Error Type	Scope
Structured Output (SO)	Missing or malformed structured fields, including labels, automation conditions, query responses, or empty outputs.
Mode Selection (MS)	Incorrectly choosing between execution and clarification.
Automation Judging (AJ)	Incorrect or failed validation of automation conditions.
Action/Tool Validity (AV/TV)	Invalid tool calls, including missing arguments, invalid `None` values, wrong names, type errors, or other schema violations.
Interaction Efficiency (IE)	Inefficient interaction behavior, especially excessive tool use.
Format/Runtime Failure (FRF)	Unparseable JSON output that prevents the evaluator from extracting executable actions.



(a) EIA Setting **(b) DR Setting**

Figure 6: Error type distributions of DeepSeek-V3.2 on DR and EIA.

4.4 Error Analysis

We analyzed the failure modes of existing models when converting smart-home intents into structured actions. As summarized in Table 3, these models exhibit diverse errors when generating executable actions. These errors span structured output (SO), mode selection (MS), automation judgment (AJ), action/tool validity (AV/TV), interaction efficiency (IE), and format/runtime robustness (FRF). Based on these six error dimensions, we analyze DeepSeek-V3.2 failure cases under the EIA and DR settings. As shown in Figure 6, SO remains the predominant error source in both settings, indicating models still struggle to follow the specified schema in constrained tasks. In the EIA setting, IE errors are notably prominent, likely because models tend to invoke redundant tools when processing ambiguous instructions and lose parameter boundaries during multi-turn interactions. In DR, mode-selection and action-validity errors are also prominent. Among the recorded mode-selection errors, unnecessary clarification occurs more frequently than execution when clarification is required. Overall, these results show that SMH-Bench evaluates more than semantic understanding, as success also requires state grounding, appropriate clarification policies, schema adherence, condition synthesis, and efficient tool interaction. Appendix E provides full distributions and case studies.

5 Conclusion

We introduced SMH-Bench, a simulator-grounded benchmark for evaluating LLM-driven agents in realistic smart-home settings. Built on HomeEnv, SMH-Bench moves beyond static instruction-to-API matching by verifying agents through executable state transitions. It contains 1,100 human-audited tasks across seven capability categories and 22 fine-grained subcategories, spanning homes from small apartments to nested multi-room environments with up to 135 devices. Our evaluation of 13 representative LLMs under both DR and EIA settings shows that current models handle explicit control and simple queries relatively well, but remain unreliable on automation, ambiguity resolution, personalized memory, and complex-home grounding. These results suggest that future smart-home agents need stronger state grounding, clarification policies, preference-aware reasoning, and schema-reliable tool use. We hope SMH-Bench will support future progress toward more reliable, context-aware, and deployable smart-home agents.

AI USE STATEMENT

We used GPT-5 to assist in constructing SMH-Bench, including generating natural-language instructions and reference assistant responses under predefined task templates and constraints. GPT-5 also served as a semantic judge for clarification requests and query responses. Generated instances were checked in HomeEnv for executability and consistency with the specified evaluation targets. Human reviewers additionally assessed instruction naturalness, category alignment, and consistency with dialogue context or user memory. We also conducted a sampled human audit of the semantic judgments. The authors take responsibility for the final dataset, evaluation protocol, analyses, and manuscript.

ETHICS STATEMENT

SMH-Bench evaluates language-model agents in simulated smart-home environments. Its Chinese-language instructions and household naming conventions may limit its representativeness across languages, cultures, and user populations. Performance on this benchmark should not be interpreted as evidence of safe or reliable end-to-end deployment in real homes. The evaluation assumes normalized device interfaces and does not cover several deployment risks, including authentication failures, state desynchronization, and physical execution failures. Smart-home agents may also introduce privacy and surveillance risks, reduce user control, or provide uneven benefits across user groups. Future deployment should therefore include appropriate privacy protections, explicit user authorization, accessible interaction mechanisms, and independent safety evaluation beyond the benchmark.

REPRODUCIBILITY STATEMENT

To support reproducibility, Appendix A specifies the HomeEnv state schema, Appendix B documents the task and instruction generation rules, and Appendix C provides the detailed task taxonomy and representative examples. Appendix D describes the Direct Reasoning and Environment-Interactive Agent evaluation settings and the shared evaluation pipeline, while Appendix H provides the prompt templates used for inference and semantic judging. Appendices E and F report detailed error analyses and model performance results. The evaluation instances and implementation prompts are in Chinese; the English prompt templates in Appendix H are translations provided for readability. The code, benchmark data, and execution instructions are available in the anonymous repository: `https://anonymous.4open.science/r/submission-code-2026-4D77/`.

REFERENCES

Hande Alemdar, Halil Ertan, Ozlem Durmaz Incel, and Cem Ersoy. Aras human activity datasets in multiple homes with multiple residents. In *2013 7th International Conference on Pervasive Computing Technologies for Healthcare and Workshops*, pp. 232–235. IEEE, 2013.

Diane J. Cook, Aaron S. Crandall, Brian L. Thomas, and Narayanan C. Krishnan. Casas: A smart home in a box. *Computer*, 46(7):62–69, 2013. doi: 10.1109/MC.2012.328. URL https://doi.org/10.1109/MC.2012.328.

Xinyu Huang, Leming Shen, Zijing Ma, and Yuanqing Zheng. Towards privacy-preserving and personalized smart homes via tailored small language models, 2025. URL https://arxiv.org/abs/2507.08878.

Jeonghye Kim, Sojeong Rhee, Minbeom Kim, Dohyung Kim, Sangmook Lee, Youngchul Sung, and Kyomin Jung. ReflAct: World-grounded decision making in LLM agents via goal-state reflection. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 33433–33465, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1697. URL https://aclanthology.org/2025.emnlp-main.1697/.

Taewoong Kim, Cheolhong Min, Byeonghwi Kim, Jinyeon Kim, Wonje Jeung, and Jonghyun Choi. Realfred: An embodied instruction following benchmark in photo-realistic environments, 2024. URL https://arxiv.org/abs/2407.18550.

Evan King, Haoxiang Yu, Sangsu Lee, and Christine Julien. Sasha: Creative goal-oriented reasoning in smart homes with large language models. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 8(1):1–38, March 2024. ISSN 2474-9567. doi: 10.1145/3643505. URL http://dx.doi.org/10.1145/3643505.

Fu Li, Jiaming Huang, Yi Gao, and Wei Dong. Chatiot: Zero-code generation of trigger-action based iot programs with chatgpt. In *Proceedings of the 7th Asia-Pacific Workshop on Networking*, pp. 219–220, 2023. doi: 10.1145/3600061.3603141. URL https://doi.org/10.1145/3600061.3603141.

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 Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12230–12250, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.597. URL https://aclanthology.org/2025.acl-long.597/.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. Toolace: Winning the points of llm function calling, 2025. URL https://arxiv.org/abs/2409.00920.

Christoph Lutz and Gemma Newlands. Privacy and smart speakers: A multi-dimensional approach. *The Information Society*, 37(3):147–162, 2021. doi: 10.1080/01972243.2021.1897914. URL https://doi.org/10.1080/01972243.2021.1897914.

Indra Mckie, Bhuva Narayan, and Baki Kocaballi. Conversational voice assistants and a case study of long-term users: A human information behaviours perspective, 2022. URL https://www.tandfonline.com/doi/full/10.1080/24750158.2022.2104738.

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. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(2): 2017–2025, 2022. doi: 10.1609/aaai.v36i2.20097. URL https://ojs.aaai.org/index.php/AAAI/article/view/20097.

Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

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/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8494–8502, 2018. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Puig_VirtualHome_Simulating_Household_CVPR_2018_paper.html.

Dmitriy Rivkin, Francois Hogan, Amal Feriani, Abhisek Konar, Adam Sigal, Steve Liu, and Greg Dudek. Sage: Smart home agent with grounded execution, 2023. URL https://arxiv.org/abs/2311.00772.

Dmitriy Rivkin, Francois Hogan, Amal Feriani, Abhisek Konar, Adam Sigal, Xue Liu, and Gregory Dudek. Aiot smart home via autonomous llm agents. *IEEE Internet of Things Journal*, 12(3): 2458–2472, 2025. doi: 10.1109/JIOT.2024.3471904.

Gyuhyeon Seo, Jungwoo Yang, Junseong Pyo, Nalim Kim, Jonggeun Lee, and Yohan Jo. Simuhome: A temporal- and environment-aware benchmark for smart home llm agents, 2026. URL https://arxiv.org/abs/2509.24282.

Yingtian Shi, Xiaoyi Liu, Chun Yu, Tianao Yang, Cheng Gao, Chen Liang, and Yuanchun Shi. Bridging the gap between natural user expression with complex automation programming in smart homes, 2024. URL https://arxiv.org/abs/2408.12687.

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*, pp. 10740–10749, 2020. URL https://openaccess.thecvf.com/content_CVPR_2020/html/Shridhar_ALFRED_A_Benchmark_for_Interpreting_Grounded_Instructions_for_Everyday_Tasks_CVPR_2020_paper.html.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. {ALFW}orld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn.

Sanjana Srivastava, Chengshu Li, Michael Lingelbach, Roberto Martín-Martín, Fei Xia, Kent Elliott Vainio, Zheng Lian, Cem Gokmen, Shyamal Buch, C. Karen Liu, Silvio Savarese, Hyowon Gweon, Jiajun Wu, and Li Fei-Fei. Behavior: Benchmark for everyday household activities in virtual, interactive, and ecological environments. In *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pp. 477–490. PMLR, 2022. URL https://proceedings.mlr.press/v164/srivastava22a.html.

Wenxuan Wang, Shi Juluan, Zixuan Ling, Yuk-Kit Chan, Chaozheng Wang, Cheryl Lee, Youliang Yuan, Jen-tse Huang, Wenxiang Jiao, and Michael R. Lyu. Learning to ask: When LLM agents meet unclear instruction. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 21773–21784, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1104. URL https://aclanthology.org/2025.emnlp-main.1104/.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 20744–20757. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/82ad13ec01f9fe44c01cb91814fd7b8c-Paper-Conference.pdf.

Ziqi Yin, Mingxin Zhang, and Daisuke Kawahara. Harmony: A human-aware, responsive, modular assistant with a locally deployed large language model, 2025. URL https://arxiv.org/abs/2410.14252.

Chaerin Yu, Chihun Choi, Sunjae Lee, Hyosu Kim, Steven Y. Ko, Young-Bae Ko, and Sangeun Oh. Leveraging llms for efficient and personalized smart home automation, 2026. URL https://arxiv.org/abs/2601.04680.

Nima Zargham, Leon Reicherts, Michael Bonfert, Sarah Theres Völkel, Johannes Schöning, Rainer Malaka, and Yvonne Rogers. Understanding circumstances for desirable proactive behaviour of voice assistants: The proactivity dilemma. In *Proceedings of the 4th Conference on Conversational User Interfaces*, Glasgow, United Kingdom, 2022. doi: 10.1145/3543829.3543834. URL https://doi.org/10.1145/3543829.3543834.

Michael J. Q. Zhang, W. Bradley Knox, and Eunsol Choi. Modeling future conversation turns to teach llms to ask clarifying questions, 2025. URL https://arxiv.org/abs/2410.13788.

Xinyi Zhao, Congjing Zhang, Pei Guo, Wei Li, Lin Chen, Chaoyue Zhao, and Shuai Huang. Smarthome-bench: A comprehensive benchmark for video anomaly detection in smart homes using multi-modal large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2025. URL https://openaccess.thecvf.com/content/CVPR2025W/VAND/html/Zhao_SmartHome-Bench_A_Comprehensive_Benchmark_for_Video_Anomaly_Detection_in_Smart_CVPRW_2025_paper.html.

Qingsong Zou, Zhi Yan, Zhiyao Xu, Kuofeng Gao, Jingyu Xiao, and Yong Jiang. Smartbench: Evaluating llms in smart homes with anomalous device states and behavioral contexts, 2026. URL https://arxiv.org/abs/2603.06636.

A SMART-HOME STATE SPACE SCHEMA IN HOMEENV

This appendix specifies the executable HomeEnv state schema used by SMH-Bench. HomeEnv represents a home through rooms and devices; devices expose typed attributes and validated services, while multi-component devices use component namespaces to disambiguate attribute and service locators. This schema is the state representation used for action replay, state-difference extraction, and rule-based verification in the benchmark.

A.1 HOME-LEVEL STATE

A HomeEnv state is stored as a structured home object with two top-level entries: `rooms` and `devices`. Rooms provide the spatial layout of the home, while devices provide executable entities assigned to rooms. Each device contains either device-level or component-level attributes and services, which together support grounding, action execution, state-difference extraction, and rule-based verification.

Table 4: Top-level objects in the HomeEnv smart-home state space.

Object	Role	Relation
Home	Executable environment state.	Contains `rooms` and `devices`.
Room	Spatial unit used for grounding.	Identified by `id`; referenced by device metadata.
Device	Controllable or observable entity.	Assigned to exactly one room through `userdata.room`.
Attribute	Typed state variable.	Belongs to a device or component.
Service	Valid state transition.	Belongs to a device or component and updates attributes after validation.
Component	Functional namespace for multi-function devices.	Groups attributes and services under paths such as `light.*`.

A.2 ROOM SCHEMA

Rooms define the spatial structure of the simulated home. Each room has a unique ID, a semantic type, a user-facing name, and a floor index. HomeEnv also supports nested room structures through the optional `parent` field, allowing environments to represent spaces such as a bathroom inside a bedroom or a balcony attached to a bedroom.

Table 5: Room schema.

Field	Type	Description
`id`	string	Unique room ID.
`type`	string	Semantic room type.
`name`	string	User-facing room name used in instructions and observations.
`floor`	integer	Floor index.
`parent`	string, optional	Parent room ID for nested spaces.

A.3 DEVICE SCHEMA

Devices connect spatial grounding with executable control. Each device is assigned to one room through `userdata.room`, exposes dynamic state through attributes, and exposes valid state transitions through services. Devices with multiple functional parts, such as fan-lights, use `components` instead of a single flat attribute-service namespace.

Table 8 summarizes representative device categories covered by HomeEnv.

Table 6: Device schema.

Field	Type	Description
`name`	string	Human-readable device name.
`description`	string	Natural-language device description.
`attributes`	list	Device-level state variables. Empty when all functions are component-level.
`services`	list	Device-level executable operations. Empty when all functions are component-level.
`components`	list	Optional component namespaces, each with attributes and services.
`userdata`	object	Static metadata for identity, retrieval, and room assignment.

Table 7: Device metadata fields in `userdata`.

Field	Type	Description
`did`	string	Unique device instance ID.
`spid`	string	Product specification ID shared by devices with the same capability schema.
`category`	string	Coarse device category.
`subcategory`	string	Fine-grained device type.
`tags`	list[string]	Semantic tags used for retrieval and filtering.
`room`	string	ID of the room where the device is located.

Table 8: Representative device categories.

Category	Representative subcategories
`light`	light, color light, color-temperature light, light strip
`sensor`	climate sensor, human sensor
`fan_light`	fan light
`media`	smart screen, universal speaker
`ac`	AC gateway, air cube, floor AC
`vacuum`	vacuum cleaner, robot cleaner
`water_heater`	electric water heater
`lock`	smart lock
`cover`	curtain, blind
`fan`	smart fan
`washing_machine`	compound washer
`oven`	built-in oven

A.4 ATTRIBUTE SCHEMA

Attributes are the basic state variables in HomeEnv. Each attribute defines a state name, type, optional value constraints, and current value. These constraints are used both during service execution and task verification. HomeEnv supports scalar types such as `str`, `bool`, `int`, `float`, `hex`, and `enum`, as well as compound types such as `list`, `tuple`, and `dict`.

Table 9: Attribute schema.

Field	Type	Description
`name`	string	Attribute name.
`description`	string	Natural-language description of the state variable.
`type`	string	Data type, such as `str`, `int`, `float`, `bool`, or `tuple`.
`value`	typed value	Current attribute value.
`range`	list, optional	Numeric lower and upper bounds.
`options`	list, optional	Finite set of allowed values.
`items`	object, optional	Element schema for compound values.
`precision`	number, optional	Numeric precision constraint.
`unit`	string, optional	Unit, such as percentage, Celsius, Kelvin, or degree.

Table 10: Representative attributes.

Device category	Attribute	Type / constraint	Meaning
General device	`state`	`str`: `on`, `off`	Power state.
General device	`battery_level`	`int`, $[0, 100]$	Battery percentage.
Light	`brightness`	`int`, $[1, 100]$	Brightness percentage.
Light	`color_temperature`	`int`, $[2700, 6500]$	Color temperature.
AC	`target_temperature`	`float`, device-specific range	Target temperature.
AC	`ac_mode`	`str`: heat, cool, auto, dry, fan_only	AC mode.
Fan	`speed_level`	`str`: low, medium, high	Fan speed level.
Sensor	`current_temperature`	`float`	Measured temperature.
Sensor	`current_humidity`	`float`	Measured humidity.
Sensor	`human_detected`	`str`: yes, no	Human-presence status.
Curtain	`position`	`int`, $[0, 100]$	Opening percentage.
Speaker	`volume_level`	`int`, $[0, 100]$	Playback volume.
Vacuum	`suction`	`str`: low, medium, high, auto	Suction level.
Lock	`alarm_volume`	`int`, $[0, 100]$	Alarm volume.

A.5 SERVICE SCHEMA

Services define executable operations that agents may call to modify the home state. A service contains a name, description, optional arguments, and an execution rule. Service arguments reuse the attribute type system, so HomeEnv can validate argument names, types, ranges, precision, and enum options before applying the state transition rule. Invalid service calls are rejected and do not mutate the home state.

Table 11: Service schema.

Field	Type	Description
`name`	string	Service name.
`description`	string	Natural-language description of the operation.
`arguments`	list	Required or optional parameters, using the same constraints as attributes.
`code`	string	State transition rule executed after validation succeeds.

Table 12: Representative services.

Device category	Service	Arguments	State update
General device	`turn_on`	none	`state = "on"`
General device	`turn_off`	none	`state = "off"`
Light	`set_brightness`	`brightness`	Updates brightness.
Light	`set_color_temperature`	`color_temperature`	Updates color temperature.
AC	`set_ac_mode`	`ac_mode`	Updates AC mode.
AC	`set_target_temperature`	`target_temperature`	Updates target temperature.
Fan	`set_speed_level`	`speed_level`	Updates fan speed.
Curtain	`open`, `close`	none	Opens or closes the curtain.
Curtain	`set_position`	`position`	Updates curtain position.
Speaker	`play`, `pause`, `stop`	metadata or none	Updates playback state.
Vacuum	`start`, `pause`, `return_to_base`	none	Updates the cleaning state of devices.
Lock	`lock`, `open`	none	Locks or unlocks the device.

A.6 Multi-Component Devices

Multi-component devices use component namespaces to prevent ambiguity between functions hosted by the same physical device. For example, a fan-light device contains a `light` component and a `fan` component. Their attributes and services are addressed through namespaced paths, such as `light.brightness` or `fan.set_speed_level`. This convention allows the evaluator to distinguish whether an action targets the light function or the fan function of the same device ID.

Table 13: Example namespace schema for a fan-light device.

Component	Attribute namespace	Service locator
`light`	`light.state`, `light.brightness`, `light.color_temperature`	`light.turn_on`, `light.set_brightness`
`fan`	`fan.state`, `fan.speed_level`, `fan.preset_mode`	`fan.turn_on`, `fan.set_speed_level`

A.7 Home-Level Example

The following valid JSON fragment illustrates how HomeEnv connects rooms, device attributes, service schemas, component namespaces, and room assignments within one executable home state. The example omits unrelated rooms and devices for brevity.

Home-level Example

```
{
  "name": "home",
  "rooms": [
    {"id": "10", "type": "dining", "name": "dining room", "
       floor": 1},
    {"id": "25", "type": "study", "name": "study room", "floor
       ": 2},
```

```
    {"id": "12", "type": "bathroom", "name": "master-bedroom
       bathroom", "floor": 1, "parent": "21"}
  ],
  "devices": [
    {
      "name": "floor-standing AC",
      "description": "floor-standing AC",
      "attributes": [
        {
          "name": "state",
          "description": "current power state of the device",
          "type": "str",
          "options": ["on", "off"],
          "value": "off"
        },
        {
          "name": "target_temperature",
          "description": "target temperature",
          "type": "float",
          "range": [18, 30],
          "precision": 1,
          "value": 18,
          "unit": "celsius"
        },
        {
          "name": "ac_mode",
          "description": "current AC mode",
          "type": "str",
          "options": ["heat", "cool", "auto", "dry", "fan_only
             "],
          "value": "heat"
        }
      ],
      "services": [
        {
          "name": "turn_on",
          "description": "turn the device on",
          "arguments": [],
          "code": "self.state = \"on\""
        },
        {
          "name": "set_target_temperature",
          "description": "set the target temperature",
          "arguments": [
            {
              "name": "target_temperature",
              "type": "float",
              "range": [18, 30],
              "precision": 1,
              "unit": "celsius"
            }
          ],
          "code": "self.target_temperature = target_temperature
             "
        }
      ],
      "components": [],
```

```
      "userdata": {
        "did": "1693",
        "spid": "100011197",
        "category": "ac",
        "subcategory": "floor_ac",
        "tags": ["ac"],
        "room": "10"
      }
    },
    {
      "name": "fan light",
      "description": "fan light",
      "attributes": [],
      "services": [],
      "components": [
        {
          "name": "light",
          "attributes": [
            {
              "name": "state",
              "description": "current power state of the light
                 component",
              "type": "str",
              "options": ["on", "off"],
              "value": "on"
            },
            {
              "name": "brightness",
              "description": "current light brightness",
              "type": "int",
              "range": [1, 100],
              "value": 58,
              "unit": "percentage"
            }
          ],
          "services": [
            {
              "name": "turn_on",
              "description": "turn the light component on",
              "arguments": [],
              "code": "self.light.state = \"on\""
            },
            {
              "name": "set_brightness",
              "description": "set the light brightness",
              "arguments": [
                {
                  "name": "brightness",
                  "description": "target light brightness",
                  "type": "int",
                  "range": [1, 100],
                  "unit": "percentage"
                }
              ],
              "code": "self.light.brightness = brightness"
            }
          ]
```

```
        },
        {
          "name": "fan",
          "attributes": [
            {
              "name": "state",
              "description": "current power state of the fan
                 component",
              "type": "str",
              "options": ["on", "off"],
              "value": "off"
            },
            {
              "name": "speed_level",
              "description": "current fan speed level",
              "type": "str",
              "options": ["low", "medium", "high"],
              "value": "medium"
            }
          ],
          "services": [
            {
              "name": "turn_on",
              "description": "turn the fan component on",
              "arguments": [],
              "code": "self.fan.state = \"on\""
            },
            {
              "name": "set_speed_level",
              "description": "set the fan speed level",
              "arguments": [
                {
                  "name": "speed_level",
                  "description": "target fan speed level",
                  "type": "str",
                  "options": ["low", "medium", "high"]
                }
              ],
              "code": "self.fan.speed_level = speed_level"
            }
          ]
        }
      ],
      "userdata": {
        "did": "3460",
        "spid": "20000010",
        "category": "fan_light",
        "subcategory": "fan_light",
        "tags": ["fan", "light"],
        "room": "25"
      }
    },
    {
      "...": "additional devices omitted"
    }
  ]
}
```

B Generation Rules and Examples for Task and Instruction Construction

This appendix details how SMH-Bench constructs user instructions and executable references from a home environment. In the implementation, Task and Instruction Generation is a *template-guided generation and grounding* procedure. A generated sample is grounded by the combination of (i) the task-specific user-generation template, (ii) the generated user instruction, (iii) the reference assistant JSON, and (iv) replay-derived labels and checks.

B.1 Template-Guided Generation and Grounding

The generation flow is organized as four sequential stages. The user-facing instruction is produced first under a task-template constraint; the executable target is then recovered through a reference assistant response and simulator replay. This design keeps the language natural while ensuring that every benchmark sample is grounded in devices, services, and states that actually exist in the home.

Stage 1: Home-state serialization. The generation flow first serializes the HomeEnv instance into a textual environment containing rooms, devices, attributes, supported services, scenes, the entry device, and optional dialogue or memory context. This serialized view is the only state evidence shown to later generation calls, so it constrains the task to valid device capabilities and prevents references to nonexistent rooms or unsupported operations.

Stage 2: User instruction generation. Given the serialized home, chat history, and one sampled user-generation template, the first LLM call writes the user side of the task. Its output contains an `analysis` field and a `user_query`; templates that deliberately create ambiguity may additionally preserve an `intended_query`. The template fixes the task family, target-selection rule, and surface style, while the LLM realizes these constraints as a realistic household utterance.

Stage 3: Reference assistant generation. The generated `user_query` is then paired with the same home state and dialogue history in a second LLM call. The reference assistant returns JSON with a `mode`, a natural-language `response`, and one of executable `actions`, `automations`, or a clarification. This step grounds the utterance back to concrete device identifiers, service locators, argument values, automation conditions, query answers, or clarification behavior.

Stage 4: Replay and label extraction. Finally, the reference trajectory is replayed against the executable HomeEnv state. Replay records the state changes caused by the reference actions and converts them into deterministic evaluation targets, including `ground_truth.labels`, `ground_truth.conditions`, and `critic`. As a result, different surface phrasings can be accepted during generation, but evaluation is performed against stable device-level outcomes.

Generation example

```
## Stage 1: Home-state evidence
Example:
The living room contains two controllable lights.
Living-room main light:
- did = 1001
- attribute = light.brightness: 85
- service = light.set_brightness(brightness: integer, range
    1..100)

Living-room downlight:
- did = 1002
- attribute = light.brightness: 90
- service = light.set_brightness(brightness: integer, range
    1..100)

The home also contains one bedroom lamp:
Bedroom lamp:
```

```
- did = 2001
- attribute = light.brightness: 35
- service = light.set_brightness(brightness: integer, range
    1..100)



## Stage 2: User instruction generation
Example output:
{
  "analysis": "The living room has two lights with brightness
      above 80, while the bedroom lamp is only 35 and is
     outside the requested room. The instruction should dim
     only the living-room lights satisfying the threshold.",
  "user_query": "Set the living-room lights above 80%
     brightness to 40%."
}



## Stage 3: Reference assistant generation
Example output:
{
  "mode": "execution",
  "response": "The living-room lights above 80% brightness
     have been set to 40%.",
  "actions": [
    {
      "did": "1001",
      "locator": "light.set_brightness",
      "arguments": {"brightness": 40}
    },
    {
      "did": "1002",
      "locator": "light.set_brightness",
      "arguments": {"brightness": 40}
    }
  ]
}



## Stage 4: Replay-derived labels
Example output:
{
  "ground_truth": {
    "labels": [
      {"did": "1001", "attribute": "light.brightness", "value":
          40},
      {"did": "1002", "attribute": "light.brightness", "value":
          40}
    ]
  },
  "critic": "device('1001').light.brightness == 40 and device
     ('1002').light.brightness == 40"
}
```

B.2 TEMPLATE RULES AND CONSTRAINTS

The first LLM call samples one task-specific template. The template determines which target pattern can be selected from the home state and how the user instruction should be expressed. Table 14 lists the complete template families used for Task and Instruction Generation.

Table 14: Template families and generation rules.

Family	Templates	Grounding rule	Surface instruction rule
`TC1: Atomic Control`	`Clear Commands`; `Colloquial Requests`; `Noisy Utterances`	Select one controllable device and one supported service or attribute change. The target must be unique and directly executable.	`clear` is explicit; `colloquial` uses everyday phrasing; `noisy` may include filler words, repetition, ASR-like artifacts, or immediate self-correction.
`TC2: Compositional Control`	`Explicit Multi-device`; `Batch Operations`; `State-dependent`; `Room-dependent`; `Top-N Selection`	Select multiple operations, a device set, a state-filtered subset, a room-conditioned subset, or a ranked top-k target. All selected devices must support the requested operation.	Express the dependency in one household request, e.g., "all lights", "lights above 80%", "rooms where the AC is on", or "the two fastest fans".
`TC3: Ambiguous Intent`	`State-level Ambiguity`; `Scene-level Ambiguity`; `Slot-missing`	Choose a plausible target but make the initial utterance under-specified, subjective, or scene-like. Missing slots may include room, device, mode, value, duration, or time.	The initial query should require clarification or context-based inference. The hidden target is preserved through `intended_query` or the reference clarification trajectory.
`TC4: Automated Task Scheduling`	`Time-specific`; `State-driven`; `Time-and-state`	Choose supported actions and attach a delay, scheduled time, state trigger, or combined time-state condition.	Use natural scheduling language such as "in two hours", "every morning at 7", "when the window opens", or "at night if the humidity is high".
`TC5: Context-Aware Multi-turn Interaction`	`Context Continuation`; `Topic Shifts`; `Coreference Resolution`; `User Corrections`	The current target depends on previous turns. The generator may inherit a device, override a previous operation, resolve a pronoun, or switch topics while preserving context.	Use natural dialogue forms such as "also do that", "no, make it 26", "turn that one off", or a topic-shifted follow-up.
`TC6: Personalized Memory`	`Short-term Memory`; `Long-term Memory`	Use recent interaction facts or stored user preferences to fill omitted action details. The task should not be solvable in the intended way without the provided memory.	Allow compact preference-triggering instructions such as "sleep time", "as usual", or "set it how I like it".
`TC7: Environment-Grounded Query`	`Status Inquiries`; `Quantity Queries`	Compute the answer from the home state, device attributes, or available tools without emitting device-control actions.	Ask a natural question about counts, locations, current status, extrema, comparisons, or device availability.

C Detailed Task Taxonomy

This appendix expands the task taxonomy. SMH-Bench contains 22 fine-grained subcategories under seven primary task categories. Each subcategory contains 50 evaluation instances in the released diverse subset, with 25 simple-home instances, 15 medium-home instances, and 10 complex-home instances. Table 15 consolidates the full taxonomy into a single table.

Table 15: Full task taxonomy with representative examples and critic-style evaluation targets.

Subcategory	Representative instruction	Evaluation target
TC1: Atomic Control.		
Clear Commands	"Set the water heater in the west kitchen to a target water temperature of 45 degrees."	**Target.** Set the west-kitchen water heater target water temperature to 45. **Critic.** `device(9470).target_water_temperature == 45`
Colloquial Requests	"Put the main-bedroom fan into blowing mode."	**Target.** Turn on the main-bedroom fan and set normal mode. **Critic.** `device(4267).state == on and device(4267).preset_mode == normal`
Noisy Utterances	"Set the bedroom temperature-humidity sensor, no, wait, the living-room sensor, alias to living-room sensor."	**Target.** Apply the corrected target, not the discarded one. **Critic.** `device(3960).alias == living-room sensor`
TC2: Compositional Control.		
Explicit Multi-device	"Resume music on the spare-bedroom smart screen and set its volume to 40; open and relock the stuck door lock; set the dining-room colored light to 30% brightness; turn on the kitchen fan in natural mode."	**Target.** Satisfy all listed final states while preserving unrelated devices. **Critic.** `device(5835).work_state == playing and volume_level == 40; device(1144). brightness == 30; device(5934).fan. state == on and fan.preset_mode == natural`
Batch Operations	"Set all lights in the home to 30% brightness and the warmest color temperature."	**Target.** Apply the same setting to every selected light. **Critic.** `all selected lights: state == on, brightness == 30, color_temperature == 2700`
State-dependent	"Set all lights whose brightness is above 80% to 40%."	**Target.** Only lights satisfying the brightness predicate are changed. **Critic.** `device(2965).light.brightness == 40; device(6460).brightness == 40; device(4072).light.brightness == 40`
Room-dependent	"Except for rooms where activity is detected, turn off the lights in the other rooms and stop any speakers or smart screens still playing there."	**Target.** Preserve occupied rooms and update only unoccupied-room devices. **Critic.** `device(6681).work_state in {standby, paused}; device(7602).state == off; device(6022).state == off`

Continued on next page

Subcategory	Representative instruction	Evaluation target
Top-N Selection	"Set the two fans with the highest wind speed to sleep mode and turn on oscillation for both."	**Target.** Select the two highest-speed fans and update both. **Critic.** `device(5035).state == on and preset_mode == sleep; device(5889).fan.state == on and fan.preset_mode == sleep and fan.oscillation_enabled == True`
TC3: Ambiguous Intent.		
State-level Ambiguity	"The room is too dark."	**Target.** Resolve or clarify the lighting intervention, then adjust the selected lights or curtains. **Critic.** `device(3460).light.brightness == 80; device(2695).position == 100; device(8066).brightness == 80`
Scene-level Ambiguity	"I am going to rest for a while."	**Target.** Ground the scene in the appropriate smart-home action. **Critic.** `device(9354).media_name == light music and volume_muted_enabled == False`
Slot-missing	"Use the oven to heat up my lunch."	**Target.** Ask for the missing duration; after the user says five minutes, start reheating. **Critic.** `device(4272).microwave_steam_oven.state == cooking; work_mode == reheating; target_cooking_time == 300; remaining_running_time == 300`
TC4: Automated Task Scheduling.		

Time-specific	"At 10 tonight, lock the parents' bedroom door and enable defense mode; set the bathroom fan-light to the dimmest warm light and lowest fan speed."	**Target.** Store the scheduled lock, light, and fan actions.
		Trigger.
		`time_cron = 0 0 22 27 6 ?  2024`
		Critic.
		`device(8993).defend_enabled == True and state == locked; device(9121).light.state == on and color_temperature == 2700 and brightness == 1 and fan.state == on`

State-driven	"When the kitchen has had no movement for fifteen minutes, automatically turn off the foyer fan-light and set the living-room smart screen volume to 30."	**Target.** Store the condition-action rule for the fan-light and smart screen.
		Trigger.
		`device(5267).no_move_duration > 900`
		Critic.
		`device(1870).light.state == off and fan.state == off; device(7813).volume_level == 30`

Time-and-state	"Every day at 2 p.m., if the kitchen temperature is still above 35 degrees, start the living-room vacuum and set the bedroom fan-light fan to high."	**Target.** Start the vacuum and set the fan to high only when both conditions hold.
		Trigger.
		`time_cron = 0 0 14 * * ?  *; device(8666).current_temperature > 35.0`
		Critic.
		`device(2194).fan.speed_level == high and device(3607).state == cleaning`

TC5: Context-Aware Multi-turn Interaction.		

Continued on next page

Subcategory	Representative instruction	Evaluation target
Context Continuation	Previous turn: rename the main-bedroom temperature-humidity sensor. Current turn: "Change its alias to main-bedroom temperature-humidity sensor."	**Target.** Resolve "its" to the previous sensor. **Critic.** `device(8431).alias == main-bedroom temperature-humidity sensor`
Topic Shifts	Previous turn: play media on a smart screen. Current turn: "Turn on the balcony fan-light in the master room, set the fan to natural mode, and set brightness to 50%."	**Target.** Do not inherit the previous media target; execute the new fan-light request. **Critic.** `device(3625).light.brightness == 50 and fan.preset_mode == natural`
Coreference Resolution	Previous turn: turn on the air cube in the entertainment room. Current turn: "Turn on the speaker too."	**Target.** Resolve the speaker to the relevant room. **Critic.** `device(8817).player.volume_muted_enabled == False and player.work_state == playing`
User Corrections	Previous turn: set the secondary-bedroom fan-light direction forward. Current turn: "Do not adjust the secondary bedroom; turn on the boy's-room fan."	**Target.** Restore the previous fan-light setting and execute the corrected target. **Critic.** `device(3024).fan.current_direction == reverse and device(7481).state == on`
TC6: Personalized Memory.		
Short-term Memory	"Do not adjust the living-room air cube after all; turn on the guest-bathroom fan instead."	**Target.** Undo the recent air-cube adjustment and apply the new fan action. **Critic.** `device(7322).state == off and target_humidity == 46 and target_temperature == 11; device(6899).fan.state == on`
Long-term Memory	"Time to sleep."	**Target.** Retrieve the user's sleep preference and apply it. **memory_list.** The user prefers the air cube to set target humidity to 50%, turn off the breathing light, and enter fall-asleep mode when they say it is time to sleep. **Critic.** `device(3415).sleep_state == fall_asleep and breathing_light_enabled == False and state == on and target_humidity == 50`
TC7: Environment-Grounded Query.		
Status Inquiries	"What are the current states of the vacuums in the secondary bedroom and guest bathroom?"	**Target.** Answer without changing any device state. **Expected answer.** Both vacuums are in standby and are not currently cleaning. **Critic.** `N/A (non-mutating query)`
Quantity Queries	"How many fan-lights currently have both the light and fan turned on?"	**Target.** Count the devices satisfying both predicates without changing state. **Expected answer.** Zero fan-lights satisfy both predicates in the sampled record. **Critic.** `N/A (non-mutating query)`

D Implementation Details for the Two Evaluation Settings

This appendix gives the exact implementation of the two evaluation formats used in SMH-Bench. Both formats start from the same HomeEnv state and are scored by the same verifier. The only difference is the interface given to the model: *Direct Reasoning* is a full-context, one-shot prediction format, while *Environment-Interactive Agent* is a partial-observation, tool-mediated interaction format.

D.1 TWO EVALUATION FORMATS

Table 16: Two SMH-Bench evaluation formats.

Interface component	Direct Reasoning	Environment-Interactive Agent
Model-facing interface		
Initial observability	**Full context.** Request, dialogue history, memory, entrance device, time, rooms, devices, current values, component paths, service schemas, arguments, ranges, enums, and deduplicated device-type schemas	**Partial context.** Request, dialogue history, memory, entrance device, time, room list, and device index. Exact states, schemas, and locators must be obtained through tools.
Runtime interface	**One-shot.** The model produces one final answer for the stage. It cannot query HomeEnv, observe execution feedback, or repair an invalid locator before replay.	**Tool loop.** The model interacts with HomeEnv through `query_device` for candidates, specs, and status, `control_device` for immediate service execution, and `create_automation` for scheduled or conditional rules. Observations are appended before the next step. The interaction budget is 10 rounds.
Model output	**Structured final JSON.** Immediate control uses `{mode, response, actions}`, where each action has `{did, locator, arguments}`. Automation uses `{mode, response, automations}`, with conditions and actions in each automation. Clarification uses `{mode, response}`. Query-only tasks use `response`.	**Tool calls plus final text.** Intermediate steps are OpenAI-style calls to `query_device` with `{what, did, room, tags, ...}`, `control_device` with `{did, locator, arguments}`, or `create_automation` with `{conditions, actions}`. The final answer is natural language after tool use stops.
Evaluator-side extraction and scoring		
Prediction extraction	**Replay.** Parse the JSON, replay `actions` in a fresh HomeEnv copy, and convert the replayed state delta into `predictions.labels`.	**State difference** Snapshot HomeEnv before and after the tool loop. The environment delta becomes `predictions.labels`; the final assistant text becomes `predictions.response`.
Main capability tested	One-shot grounding from complete serialized context: target selection, component resolution, action formatting, argument typing, and no-op avoidance.	Interactive deployment behavior under partial observability: evidence gathering, schema discovery, tool sequencing, execution feedback handling, and stopping after the environment satisfies the request.
Same final verifier	After either format, SMH-Bench applies the same record-level checks: critic evaluation for state-changing tasks, query-response judging for query tasks, clarification alignment for clarification tasks, token accounting, and final `overall_pass`.	

D.2 EVALUATION PIPELINE

SMH-Bench uses a unified evaluation pipeline for both settings. Given a set of benchmark samples, the pipeline either loads existing predictions or runs model inference under one of the two interfaces.

In the Direct Reasoning setting, the model receives the serialized home context and produces a structured final prediction, which is replayed in a fresh HomeEnv instance to obtain the predicted state changes. In the Environment-Interactive Agent setting, the model observes a partial home context, interacts with HomeEnv through tools, and the resulting environment delta is used as the prediction.

After inference, both settings are normalized into the same prediction schema, including predicted state labels, optional natural-language responses, automation conditions, dialogue traces, token usage, and execution errors. The shared verifier then evaluates each sample according to its task type: state-changing tasks are checked by the critic over the predicted home state, query-only tasks are judged by response quality, clarification tasks require the appropriate clarification behavior, and automation tasks additionally check predicted trigger or condition semantics. A sample passes only when all required checks for that sample type pass. Finally, SMH-Bench reports record-level evaluation results and aggregate pass rates by task category and evaluation setting.

D.3 Direct Reasoning Trajectory and Environment-Interactive Agent Trajectory

Direct Reasoning Trajectory

```
Query:
  Turn on all kitchen-area lights that are currently off, and
      set their brightness to 70%.

Observation:
  The model receives full home context before generation.
  Relevant excerpt:
    room 11 = kitchen area
    did 9313 = fan-light, tags=[fan, light], room=11
      state: light.state=off, light.brightness=32
      services: light.turn_on(), light.set_brightness(
          brightness:int[1,100])
    did 6152 = light, tags=[light], room=11
      state: state=off, brightness=33
      services: turn_on(), set_brightness(brightness:int
          [1,100])
    did 2284 = light, tags=[light], room=11
      state: state=off, brightness=16
      services: turn_on(), set_brightness(brightness:int
          [1,100])
    did 8381 = light, tags=[light], room=11
      state: state=off, brightness=16
      services: turn_on(), set_brightness(brightness:int
          [1,100])

Example Trajectory:
  Step 1. The model directly emits the final structured
      execution payload:
  {
    "mode": "execution",
    "response": "The kitchen-area lights that were off are now
        on and set to 70% brightness.",
    "actions": [
      {"did": "9313", "locator": "light.turn_on", "arguments":
          {}},
      {"did": "9313", "locator": "light.set_brightness", "
          arguments": {"brightness": 70}},
      {"did": "6152", "locator": "turn_on", "arguments": {}},
```

```
    {"did": "6152", "locator": "set_brightness", "arguments":
        {"brightness": 70}},
    {"did": "2284", "locator": "turn_on", "arguments": {}},
    {"did": "2284", "locator": "set_brightness", "arguments":
        {"brightness": 70}},
    {"did": "8381", "locator": "turn_on", "arguments": {}},
    {"did": "8381", "locator": "set_brightness", "arguments":
        {"brightness": 70}}
  ]
 }

Replay:
 The evaluator starts from a fresh copy of engine_0 and
    executes the predicted actions.
  action 1: 9313.light.turn_on() -> 9313.light.state: off ->
      on
  action 2: 9313.light.set_brightness(70) -> 9313.light.
     brightness: 32 -> 70
  action 3: 6152.turn_on() -> 6152.state: off -> on
  action 4: 6152.set_brightness(70) -> 6152.brightness: 33
     -> 70
  action 5: 2284.turn_on() -> 2284.state: off -> on
  action 6: 2284.set_brightness(70) -> 2284.brightness: 16
     -> 70
  action 7: 8381.turn_on() -> 8381.state: off -> on
  action 8: 8381.set_brightness(70) -> 8381.brightness: 16
     -> 70

Prediction:
 labels = [
  {"did":"9313","attribute":"light.state","value":"on"},
  {"did":"9313","attribute":"light.brightness","value":70},
  {"did":"6152","attribute":"state","value":"on"},
  {"did":"6152","attribute":"brightness","value":70},
  {"did":"2284","attribute":"state","value":"on"},
  {"did":"2284","attribute":"brightness","value":70},
  {"did":"8381","attribute":"state","value":"on"},
  {"did":"8381","attribute":"brightness","value":70}
 ]

Verification:
 Critic =
  device('9313').light.state == "on" and device('9313').
     light.brightness == 70 and
  device('6152').state == "on" and device('6152').brightness
      == 70 and
  device('2284').state == "on" and device('2284').brightness
      == 70 and
  device('8381').state == "on" and device('8381').brightness
      == 70
 Result: TRUE. The sample passes under Direct Reasoning.
```

Environment-Interactive Agent Trajectory

```
Query:
```

```
  Turn on all kitchen-area lights that are currently off, and
      set their brightness to 70%.

Observation:
  The model receives only brief home context before acting.
  Relevant excerpt:
    room 11 = kitchen area
    device index in room 11:
      did 9313 = fan-light, tags=[fan, light]
      did 6152 = light, tags=[light]
      did 2284 = light, tags=[light]
      did 8381 = light, tags=[light]
  Exact attributes, service locators, and argument schemas
      are not assumed; they are queried.

Example Trajectory:
  Round 1. query_device
    query_device({"what":"spec_status","did":"9313"})
    query_device({"what":"spec_status","did":"6152"})
    query_device({"what":"spec_status","did":"2284"})
    query_device({"what":"spec_status","did":"8381"})

    Observations:
      9313 -> fan-light; light.state=off; light.brightness=32;
            valid light services: light.turn_on(),
            light.set_brightness(brightness:int[1,100]).
      6152 -> device-level light; state=off; brightness=33;
            valid services: turn_on(), set_brightness(
               brightness:int[1,100]).
      2284 -> device-level light; state=off; brightness=16;
            valid services: turn_on(), set_brightness(
               brightness:int[1,100]).
      8381 -> device-level light; state=off; brightness=16;
            valid services: turn_on(), set_brightness(
               brightness:int[1,100]).

  Round 2. control_device
    control_device({"did":"9313","locator":"light.turn_on","
       arguments":{}})
    control_device({"did":"9313","locator":"light.
       set_brightness","arguments":{"brightness":70}})
    control_device({"did":"6152","locator":"turn_on","
       arguments":{}})
    control_device({"did":"6152","locator":"set_brightness","
       arguments":{"brightness":70}})
    control_device({"did":"2284","locator":"turn_on","
       arguments":{}})
    control_device({"did":"2284","locator":"set_brightness","
       arguments":{"brightness":70}})
    control_device({"did":"8381","locator":"turn_on","
       arguments":{}})
    control_device({"did":"8381","locator":"set_brightness","
       arguments":{"brightness":70}})

    Observations:
      ok=true; 9313.light.state: off -> on.
      ok=true; 9313.light.brightness: 32 -> 70.
```

```
      ok=true; 6152.state: off -> on.
      ok=true; 6152.brightness: 33 -> 70.
      ok=true; 2284.state: off -> on.
      ok=true; 2284.brightness: 16 -> 70.
      ok=true; 8381.state: off -> on.
      ok=true; 8381.brightness: 16 -> 70.

  Final response:
    "Done. I turned on the kitchen-area lights that were off
       and set them to 70% brightness."

Prediction:
  The evaluator does not parse labels from the final text.
  It computes labels by diffing the HomeEnv snapshot before
     Round 1 and after Round 2:
    {"did":"9313","attribute":"light.state","value":"on"}
    {"did":"9313","attribute":"light.brightness","value":70}
    {"did":"6152","attribute":"state","value":"on"}
    {"did":"6152","attribute":"brightness","value":70}
    {"did":"2284","attribute":"state","value":"on"}
    {"did":"2284","attribute":"brightness","value":70}
    {"did":"8381","attribute":"state","value":"on"}
    {"did":"8381","attribute":"brightness","value":70}

Verification:
  The same critic used above evaluates to TRUE.
  Result: TRUE. The sample passes under Environment-
     Interactive Agent.
```

E Error Distribution Breakdown

E.1 Complete Error Distribution

Table 17 reports the full error breakdown for the four representative models analyzed in the main experiments. Counts are grouped by evaluation format: Environment-Interactive Agent (EIA) and Direct Reasoning (DR). Each cell reports the number of categorized errors, with the percentage share within that model and setting in parentheses. A dash indicates that the error type was not observed in the categorized failures for that model.

Table 17: Complete breakdown of categorized error types and per-model frequencies. Percentages are computed within each model-setting error distribution.

Setting	Error type	Gemini-3.1-Pro	Claude-Sonnet-4.6	GPT-5.4	DeepSeek-V3.2
Environment-Interactive Agent (EIA)					
Structured output	Missing labels	44 (30.8%)	53 (37.6%)	116 (59.2%)	113 (43.6%)
Structured output	Missing automation conditions	13 (9.1%)	6 (4.3%)	44 (22.4%)	17 (6.6%)
Structured output	Missing query response	–	–	–	2 (0.8%)
Structured output	Empty model output	–	–	5 (2.6%)	–
Automation judging	Conditions judge error	18 (12.6%)	15 (10.6%)	18 (9.2%)	14 (5.4%)

Continued on next page

Setting	Error type	Gemini-3.1-Pro	Claude-Sonnet-4.6	GPT-5.4	DeepSeek-V3.2
Tool-call validity	Missing required tool arguments	33 (23.1%)	56 (39.7%)	9 (4.6%)	16 (6.2%)
Tool-call validity	Invalid `None` value in tool arguments	32 (22.4%)	8 (5.7%)	–	6 (2.3%)
Tool-call validity	Integer passed where string was required	–	–	–	11 (4.2%)
Tool-call validity	Other tool or schema errors	3 (2.1%)	3 (2.1%)	4 (2.0%)	5 (1.9%)
Interaction efficiency	Too many tool calls	–	–	–	75 (29.0%)
Direct Reasoning (DR)					
Structured output	Missing labels	50 (37.3%)	75 (30.0%)	96 (44.7%)	135 (34.3%)
Structured output	Missing automation conditions	12 (9.0%)	10 (4.0%)	26 (12.1%)	20 (5.1%)
Structured output	Missing query response	–	8 (3.2%)	–	–
Mode selection	Execution predicted as clarification	33 (24.6%)	14 (5.6%)	60 (27.9%)	85 (21.6%)
Mode selection	Clarification predicted as execution	32 (23.9%)	46 (18.4%)	25 (11.6%)	26 (6.6%)
Automation judging	Conditions judge error	4 (3.0%)	4 (1.6%)	4 (1.9%)	7 (1.8%)
Action/schema validity	Missing required action argument	2 (1.5%)	2 (0.8%)	2 (0.9%)	–
Action/schema validity	Invalid `None` value in action arguments	–	33 (13.2%)	–	–
Action/schema validity	Wrong action argument name	–	–	–	73 (18.5%)
Action/schema validity	Other action execution error	–	–	–	24 (6.1%)
Format/runtime	LLM request or JSON parsing error	1 (0.7%)	58 (23.2%)	2 (0.9%)	24 (6.1%)

E.2 Representative Error Case Studies

Table 18 summarizes three representative failures that correspond to the dominant categories in Table 17. We report each case by its task requirement, observed behavior, evaluator-visible failure, and underlying mechanism. This format preserves the diagnostic evidence from the recorded runs while omitting identifiers and low-level trace fields that do not affect the interpretation.

Case 1: Schema-grounding failure in single-device control. The user asked the agent to start heating the electric water heater in the secondary-bedroom public bathroom. The agent resolved the correct device, device 3981, and inspected the service schema for `start_heating`, which required `work_mode`, `target_water_quantity`, and `target_water_temperature`. However, the subsequent control call supplied only `work_mode=quick`. The environment rejected the call with a missing-argument error, and the final prediction contained no state-change labels. This case shows a schema-grounding failure: the model identified the right device and operation but failed to instantiate the complete executable signature.

Case 2: Automation serialization failure. The user requested an automation that turns on stair and corridor lights at half brightness whenever human-presence sensors in the specified areas detect a person. The model recognized the relevant trigger family and repeatedly inspected candidate sensors, but it stopped by asking for confirmation about partial sensor availability. It never issued a `create_automation` call and emitted neither `conditions` nor `labels`, which made both the trigger and effect unevaluable. This case illustrates that automation errors can occur even when the high-level intent is understood: the missing step is the serialization of the rule into concrete condition and action artifacts.

Case 3: Interaction-control failure under a tool budget. In an ambiguous hosting-scene task, the user clarified that the reception room should feel warm and welcoming by adjusting lights, air conditioning, and music. After finding four reception-room lights, the model spent six additional

Table 18: Representative failure mechanisms observed in the categorized errors.

Case	Task requirement	Observed behavior	Diagnosis
Schema grounding	Start a water heater using a service with required arguments.	The model selected the correct device and service but called `start_heating` with only `work_mode`.	Correct semantic grounding, incomplete service-signature instantiation.
Automation serialization	Create a presence-triggered automation that turns on stair and corridor lights at half brightness.	The model inspected candidate sensors but produced no `create_automation` call, no conditions, and no action labels.	High-level automation intent recognized, but not serialized into executable artifacts.
Interaction control	Configure a reception-room hosting scene under a 10-round interaction budget.	The model over-queried devices and exhausted the budget after only partial lighting changes.	Inefficient tool policy left the multi-device request incomplete.

calls inspecting individual devices before executing any action. It then completed only one light's state, brightness, and color-temperature changes before exceeding the 10-round budget, leaving the climate and music subgoals unfinished. This case is primarily an interaction-control failure rather than a semantic failure: the model understood the scene goal, but its tool-use policy consumed the budget before the multi-device task was complete.

F Detailed Model Performance Results

This appendix reports the complete success-rate breakdowns underlying the main experimental table. DR denotes the Direct Reasoning setting, and EIA denotes the Environment-Interactive Agent setting. TC1–TC7 follow the taxonomy in Section 3.3.1.

F.1 Full Pass Rates by Difficulty

Table 19 expands Table 2 by retaining the home-complexity split. Each capability cell reports success rate in percent.

Table 19: Complete final pass rates by model, evaluation setting, home difficulty, and task category.

Model	Setting	Difficulty	Avg	TC1 Atom	TC2 Comp.	TC3 Ambig.	TC4 Auto.	TC5 Multi.	TC6 Memory	TC7 Query
Qwen3.5-4B	DR	Simple	55.1	73.3	48.0	60.0	34.7	53.0	46.0	82.0
	DR	Medium	43.6	73.3	30.7	33.3	17.8	51.7	33.3	80.0
	DR	Complex	26.4	50.0	12.0	23.3	6.7	30.0	30.0	50.0
	EIA	Simple	65.8	77.3	60.0	85.3	25.3	72.0	64.0	84.0
	EIA	Medium	48.8	68.9	37.3	64.4	20.0	51.7	43.3	66.7
	EIA	Complex	44.5	50.0	22.0	50.0	13.3	67.5	60.0	70.0
Qwen3.5-9B	DR	Simple	59.8	82.7	52.8	60.0	34.7	56.0	60.0	88.0
	DR	Medium	41.8	66.7	29.3	31.1	20.0	50.0	33.3	76.7
	DR	Complex	33.6	60.0	14.0	40.0	10.0	42.5	35.0	50.0
	EIA	Simple	70.0	78.7	67.2	84.0	46.7	74.0	60.0	80.0
	EIA	Medium	62.1	71.1	52.0	64.4	24.4	80.0	63.3	90.0
	EIA	Complex	53.2	80.0	34.0	40.0	26.7	65.0	80.0	70.0
Qwen3.5-397B	DR	Simple	84.2	94.7	88.0	85.3	66.7	86.0	72.0	92.0
	DR	Medium	73.0	93.3	64.0	62.2	57.8	83.3	76.7	80.0
	DR	Complex	67.7	86.7	44.0	76.7	46.7	85.0	80.0	70.0
	EIA	Simple	84.9	93.3	83.2	90.7	72.0	87.0	74.0	94.0
	EIA	Medium	79.4	86.7	62.7	82.2	75.6	83.3	83.3	100.0
	EIA	Complex	57.7	76.7	42.0	83.3	53.3	30.0	75.0	75.0
Qwen3.5-Plus	DR	Simple	75.1	92.0	71.2	72.0	64.0	79.0	66.0	82.0
	DR	Medium	53.3	64.4	44.0	28.9	42.2	65.0	76.7	66.7

Continued on next page

Model	Setting	Difficulty	Avg	TC1 Atom	TC2 Comp.	TC3 Ambig.	TC4 Auto.	TC5 Multi.	TC6 Memory	TC7 Query
	DR	Complex	45.0	70.0	14.0	33.3	26.7	75.0	75.0	40.0
	EIA	Simple	76.9	92.0	75.2	85.3	53.3	80.0	60.0	92.0
	EIA	Medium	67.6	95.6	49.3	71.1	33.3	78.3	80.0	83.3
	EIA	Complex	56.4	83.3	24.0	63.3	30.0	80.0	55.0	80.0
DeepSeek-V3.2	DR	Simple	79.8	85.3	81.6	78.7	61.3	85.0	70.0	96.0
	DR	Medium	62.7	71.1	45.3	55.6	53.3	76.7	63.3	90.0
	DR	Complex	53.6	76.7	32.0	46.7	26.7	72.5	70.0	70.0
	EIA	Simple	85.6	90.7	80.8	98.7	74.7	88.0	74.0	94.0
	EIA	Medium	68.8	84.4	32.0	77.8	62.2	83.3	73.3	100.0
	EIA	Complex	57.7	83.3	14.0	63.3	40.0	80.0	75.0	85.0
MiniMax-M2.7	DR	Simple	64.0	88.0	76.8	74.7	52.0	39.0	32.0	80.0
	DR	Medium	45.8	77.8	44.0	51.1	28.9	28.3	40.0	60.0
	DR	Complex	30.5	70.0	20.0	40.0	20.0	22.5	20.0	25.0
	EIA	Simple	45.8	64.0	17.6	65.3	52.0	34.0	22.0	98.0
	EIA	Medium	45.2	42.2	37.3	40.0	42.2	48.3	26.7	93.3
	EIA	Complex	38.2	63.3	18.0	36.7	16.7	47.5	40.0	65.0
GLM-5	DR	Simple	72.9	90.7	72.0	77.3	68.0	71.0	34.0	92.0
	DR	Medium	50.9	75.6	38.7	20.0	40.0	68.3	46.7	76.7
	DR	Complex	40.0	63.3	12.0	23.3	30.0	75.0	40.0	45.0
	EIA	Simple	76.9	94.7	78.4	92.0	52.0	76.0	46.0	94.0
	EIA	Medium	67.3	84.4	62.7	82.2	31.1	70.0	60.0	86.7
	EIA	Complex	59.5	80.0	38.0	63.3	36.7	85.0	50.0	70.0
GPT-5.4-Mini	DR	Simple	77.3	89.3	81.6	77.3	53.3	81.0	58.0	96.0
	DR	Medium	63.9	71.1	57.3	48.9	42.2	81.7	66.7	86.7
	DR	Complex	54.1	73.3	40.0	40.0	30.0	70.0	65.0	75.0
	EIA	Simple	81.8	90.7	83.2	93.3	60.0	83.0	68.0	92.0
	EIA	Medium	74.5	91.1	60.0	84.4	55.6	80.0	70.0	93.3
	EIA	Complex	60.0	83.3	46.0	70.0	20.0	77.5	70.0	60.0
GPT-5.4	DR	Simple	85.6	92.0	90.4	84.0	73.3	89.0	66.0	98.0
	DR	Medium	77.3	93.3	68.0	64.4	66.7	83.3	80.0	96.7
	DR	Complex	74.5	86.7	66.0	83.3	43.3	80.0	90.0	85.0
	EIA	Simple	85.1	94.7	88.8	92.0	56.0	91.0	70.0	98.0
	EIA	Medium	76.1	93.3	69.3	77.8	46.7	83.3	80.0	90.0
	EIA	Complex	70.9	93.3	64.0	76.7	36.7	80.0	75.0	75.0
Gemini-3.1-Flash	DR	Simple	78.5	90.7	86.4	66.7	70.7	74.0	62.0	96.0
	DR	Medium	70.6	88.9	61.3	57.8	62.2	70.0	76.7	93.3
	DR	Complex	67.7	86.7	60.0	63.3	43.3	72.5	85.0	75.0
	EIA	Simple	81.8	89.3	85.6	90.7	56.0	82.0	70.0	98.0
	EIA	Medium	74.2	84.4	62.7	84.4	48.9	81.7	73.3	96.7
	EIA	Complex	62.3	83.3	40.0	70.0	30.0	72.5	80.0	85.0
Gemini-3.1-Pro	DR	Simple	88.9	96.0	94.4	85.3	77.3	92.0	76.0	94.0
	DR	Medium	78.8	95.6	69.3	53.3	68.9	83.3	100.0	100.0
	DR	Complex	76.8	93.3	66.0	70.0	56.7	82.5	100.0	85.0
	EIA	Simple	90.2	96.0	95.2	98.7	69.3	93.0	78.0	94.0
	EIA	Medium	81.2	91.1	73.3	86.7	60.0	83.3	96.7	90.0
	EIA	Complex	78.6	93.3	74.0	83.3	56.7	85.0	90.0	70.0
Claude-Haiku-4.5	DR	Simple	84.0	93.3	88.0	81.3	76.0	85.0	76.0	82.0
	DR	Medium	71.5	86.7	61.3	64.4	60.0	76.7	80.0	83.3
	DR	Complex	65.9	86.7	54.0	63.3	40.0	82.5	75.0	65.0
	EIA	Simple	81.8	89.3	87.2	92.0	54.7	82.0	68.0	96.0
	EIA	Medium	73.9	86.7	65.3	80.0	44.4	81.7	80.0	90.0
	EIA	Complex	65.5	86.7	52.0	76.7	36.7	77.5	75.0	60.0
Claude-Sonnet-4.6	DR	Simple	82.2	90.7	86.4	76.0	73.3	84.0	70.0	90.0
	DR	Medium	73.0	91.1	61.3	44.4	71.1	85.0	86.7	83.3
	DR	Complex	68.6	90.0	58.0	50.0	53.3	82.5	80.0	75.0
	EIA	Simple	88.0	94.7	93.6	93.3	72.0	86.0	74.0	98.0
	EIA	Medium	81.2	93.3	70.7	86.7	68.9	83.3	86.7	90.0
	EIA	Complex	78.6	90.0	66.0	93.3	60.0	85.0	85.0	80.0

F.2 DeepSeek-V3.2 Repeated Runs

To assess run-to-run variability, we evaluate DeepSeek-V3.2 under DR and EIA across five paired runs on the same task set. As shown in Table 20, EIA outperforms DR in all five runs, with a mean

improvement of 5.8 percentage points and a sample standard deviation of 1.8 points. These results support the consistency of the overall EIA advantage for DeepSeek-V3.2 under the evaluated setting.

Table 20: DeepSeek-V3.2 results across five runs. The final row reports mean ± sample SD. CIs are task-level paired-bootstrap intervals for EIA−DR.

Run	DR (%)	EIA (%)	EIA−DR (pp)	95% CI (pp)
1	69.5	75.0	$+5.5$	$[+3.1, +8.0]$
2	71.1	74.6	$+3.5$	$[+0.7, +6.3]$
3	68.9	77.1	$+8.2$	$[+5.3, +11.1]$
4	72.0	77.0	$+5.0$	$[+2.2, +7.8]$
5	69.8	76.4	$+6.6$	$[+3.7, +9.4]$
Mean ± SD	70.3 ± 1.3	76.0 ± 1.2	$+5.8 \pm 1.8$	—

F.3 DeepSeek-V3.2 Thinking Ablation

Table 21 reports the complete DeepSeek-V3.2 ablation between the thinking run and the non-thinking run.Diff is computed as Non-Thinking minus Thinking, so negative values indicate degradation after removing thinking.

Table 21: DeepSeek-V3.2 thinking versus non-thinking success rates by evaluation setting, difficulty, and task category.

Difficulty	Category	Thinking	Non-thinking	Diff
EIA				
Simple	Ambiguous Intent	0.9867	0.8667	-0.1200
Simple	Atomic Control	0.9067	0.7333	-0.1733
Simple	Automated Task Scheduling	0.7467	0.6667	-0.0800
Simple	Compositional Control	0.8080	0.7280	-0.0800
Simple	Personalized Memory	0.7400	0.4800	**-0.2600**
Simple	Context-Aware Multi-turn Interaction	0.8800	0.7600	-0.1200
Simple	Environment-Grounded Query	0.9400	0.9200	-0.0200
Medium	Ambiguous Intent	0.7778	0.8000	+0.0222
Medium	Atomic Control	0.8444	0.8667	+0.0222
Medium	Automated Task Scheduling	0.6222	0.5778	-0.0444
Medium	Compositional Control	0.3200	0.2933	-0.0267
Medium	Personalized Memory	0.7333	0.8333	+0.1000
Medium	Context-Aware Multi-turn Interaction	0.8333	0.7000	-0.1333
Medium	Environment-Grounded Query	1.0000	0.7667	**-0.2333**
Complex	Ambiguous Intent	0.6333	0.6000	-0.0333
Complex	Atomic Control	0.8333	0.9000	+0.0667
Complex	Automated Task Scheduling	0.4000	0.5000	+0.1000
Complex	Compositional Control	0.1400	0.1600	+0.0200
Complex	Personalized Memory	0.7500	0.5000	**-0.2500**
Complex	Context-Aware Multi-turn Interaction	0.8000	0.8500	+0.0500
Complex	Environment-Grounded Query	0.8500	0.6500	-0.2000
DR				
Simple	Ambiguous Intent	0.7867	0.7733	-0.0133
Simple	Atomic Control	0.8533	0.7333	-0.1200
Simple	Automated Task Scheduling	0.6133	0.4933	-0.1200
Simple	Compositional Control	0.8160	0.6000	-0.2160
Simple	Personalized Memory	0.7000	0.4000	**-0.3000**
Simple	Context-Aware Multi-turn Interaction	0.8500	0.7000	-0.1500
Simple	Environment-Grounded Query	0.9600	0.7800	-0.1800
Medium	Ambiguous Intent	0.5556	0.4000	-0.1556
Medium	Atomic Control	0.7111	0.5333	-0.1778
Medium	Automated Task Scheduling	0.5333	0.2889	-0.2444
Medium	Compositional Control	0.4533	0.3067	-0.1467
Medium	Personalized Memory	0.6333	0.3333	**-0.3000**
Medium	Context-Aware Multi-turn Interaction	0.7667	0.5167	-0.2500
Medium	Environment-Grounded Query	0.9000	0.6667	-0.2333
Complex	Ambiguous Intent	0.4667	0.2667	-0.2000

Continued on next page

Difficulty	Category	Thinking	Non-thinking	Diff
Complex	Atomic Control	0.7667	0.4333	**-0.3333**
Complex	Automated Task Scheduling	0.2667	0.3000	+0.0333
Complex	Compositional Control	0.3200	0.0800	-0.2400
Complex	Personalized Memory	0.7000	0.3500	**-0.3500**
Complex	Context-Aware Multi-turn Interaction	0.7250	0.5000	-0.2250
Complex	Environment-Grounded Query	0.7000	0.4500	-0.2500

G HUMAN AUDIT AND JUDGE VALIDATION

Construction audit. A pool of five annotators conducted the construction audit. Each candidate instance was independently reviewed by three annotators for instruction naturalness, category alignment, and consistency with dialogue context or user memory. Inclusion required unanimous approval from all three reviewers. In the first review round, 23% of candidate instances did not receive unanimous approval and were revised and re-reviewed or removed.

Human–judge agreement and independent-family rejudging. The construction audit was separate from the human validation of semantic judgments. The latter used a random sample of 200 instances and yielded 98.0% agreement between human judgments and GPT-5. Separately, DeepSeek-V4-Pro rejudged all TC3/TC7 outputs and achieved 96.4% agreement with GPT-5.

Stratified analysis across evaluated model families and evaluation settings. To examine whether judging outcomes vary with the evaluated model family or evaluation setting, we stratified the rejudging results by model family and, separately, by the DR/EIA settings.

Table 22: Stratified comparison of GPT-5 and DeepSeek-V4-Pro judgments on TC3/TC7 outputs. The two judge columns report pass rates; Delta is DeepSeek-V4-Pro minus GPT-5 in percentage points.

Group	GPT-5 Judge	DeepSeek-V4-Pro Judge	Delta	Agreement
GPT-family model outputs	90.2%	89.9%	−0.3 pp	96.7%
Non-GPT-family model outputs	84.1%	83.1%	−1.0 pp	96.4%
DR	82.1%	80.6%	−1.5 pp	96.8%
EIA	88.0%	87.8%	−0.2 pp	96.0%

Delta denotes the pass rate assigned by DeepSeek-V4-Pro minus that assigned by GPT-5 on the same outputs. Agreement denotes the fraction of matching judgments. The model-family rows and the evaluation-setting rows represent two separate partitions of the evaluated outputs.

Replacing GPT-5 with the independent-family judge reduced the pass rate by 0.3 percentage points for GPT-family outputs and by 1.0 percentage point for non-GPT-family outputs. Thus, GPT-family outputs did not show a larger decrease under judge replacement. Agreement also remained high in both DR and EIA. These results provide no evidence of systematic same-family preference in the evaluated TC3/TC7 judgments.

H PROMPT TEMPLATES

This appendix lists the prompt templates used in the two evaluation settings and in the LLM-based judges. The templates are translated into English from the implementation prompts. Placeholders enclosed in braces or double braces are filled by the evaluator at runtime.

H.1 DIRECT REASONING SETTING

DR setting prompt template

```
You are a smart-home assistant. Given the current home
   environment, dialogue history, and user request, you must
   output a JSON object as the response.

Output JSON only. Do not output any additional text.

## Basic Requirements

- The response mode must be either clarification or execution.

- If the context allows you to infer a unique and executable
   target device, function, and parameter, execute directly.
- If there are multiple reasonable interpretations, or if a
   key target, function, or direction is missing, ask a
   clarification question using clarification.
- Unless the user explicitly uses universal terms such as "
   whole home", "entire house", or "all", do not assume that
   the operation applies to all devices of the same type in
   the home.
- Home Environment, the user-instruction entry point,
   dialogue history, and long-term memory are only reasoning
   cues. Whether to execute depends on whether a unique
   solution can be determined.

## Mode 1: Clarification

When no unique executable action can be determined, output:

{
  "mode": "clarification",
  "response": "A concise and specific question"
}

Example:

{
  "mode": "clarification",
  "response": "There are both a fan and an air conditioner in
       the living room that can cool the room. Which one would
       you like to adjust?"
}

## Mode 2: Execution

When the user's intent is sufficiently clear, output:

- Use actions for direct device control.
- Use automations for scheduled or conditional rules.
- Choose exactly one of actions and automations; do not
   output both.

### 2.1 Direct Control: actions
```

```
Format:

{
  "mode": "execution",
  "response": "Confirmation message for the user",
  "actions": [
    {
      "did": "device id",
      "locator": "service or component.service",
      "arguments": {}
    }
  ]
}

Core constraints:

- Before generating actions, first check whether the target
    state is already satisfied. If it is already satisfied,
    return an empty array and explain this in response; do not
     generate redundant operations.
- Values in arguments must match the parameter types defined
    in Home Environment.
- If a device must be turned on before the requested
    operation can be applied, include the power-on action
    first.
- If no matching device is found, or if the device does not
    support the required function, return empty actions and
    explain this in response.
- locator rule: if a component exists, write <component>.<
    service>; otherwise write the service directly.

Example:

{
  "mode": "execution",
  "response": "Done. I raised the bedroom air conditioner's
      temperature.",
  "actions": [
    {
      "did": "7118",
      "locator": "set_target_temperature",
      "arguments": { "temperature": 27 }
    }
  ]
}

### 2.2 Automation: automations

Format:

{
  "mode": "execution",
  "response": "Confirmation message for the user",
  "automations": [
    {
      "conditions": {
        "time_cron": "...",
```

```
      "state_expr": "..."
     },
     "actions": [
      {
       "did": "device id",
       "locator": "service or component.service",
       "arguments": {}
      }
     ]
    }
  ]
}

Core constraints:

- conditions must always contain both time_cron and
    state_expr; write null when a field is absent.
- Time conditions use Quartz seven-field cron: second minute
    hour day-of-month month day-of-week year.
- State conditions use expressions such as device('1001').
    state == "off".
- Do not write natural language inside conditions.
- If the user describes multiple independent trigger times,
    split them into multiple automations.
- If no matching device is found, or if the required function
     is unsupported, return empty automations and explain this
     in response.

Example:

{
  "mode": "execution",
  "response": "Done. I set an automation to turn off the
      living-room light every day at 10 p.m.",
  "automations": [
    {
     "conditions": {
      "time_cron": "0 0 22 * * ? *",
      "state_expr": null
     },
     "actions": [
      {
       "did": "1001",
       "locator": "turn_off",
       "arguments": {}
      }
     ]
    }
  ]
}

## Home Environment

{{home_environment}}

## User Memory
```

```
{{memory_list}}

Strictly follow the requirements above and output JSON.
```

H.2 Environment-Interactive Agent Setting

EIA setting prompt template

```
You are a smart-home assistant. Given the current home
   environment, dialogue history, and user request, you must
   understand the user's intent and, when needed, use tools
   to query devices, control devices, or create automations.

Your final output must be natural language shown to the user.
    Do not output JSON, do not output code, and do not expose
    internal reasoning.

## Basic Requirements

- If the context allows you to infer a unique and executable
   target device, function, and parameter, execute directly.
- If there are multiple reasonable interpretations, or if a
   key target, function, or direction is missing, directly
   reply with one clarification sentence.
- Unless the user explicitly uses universal terms such as "
   whole home", "entire house", or "all", do not assume that
   the operation applies to all devices of the same type in
   the home.
- Home Environment, the user-instruction entry point,
   dialogue history, and long-term memory are only reasoning
   cues. Whether to execute depends on whether a unique
   solution can be determined.

## Tool Selection

### 1. query_device

Use this tool to find devices or obtain information required
   for execution.

- brief: find candidate devices.
- spec: query device capabilities, service names, parameter
   types, and parameter ranges.
- status: query the current status to avoid redundant
   operations.
- spec_status: use this when a single device has already been
    identified and both capabilities and status are needed.

### 2. control_device

Use this tool for immediate control of a confirmed device.

- Call it only when the device, locator, and arguments are
   all clear.
- locator is either service or component.service.
```

```
- arguments must exactly match the parameters required by the
    service; pass {} for services without parameters.
- For devices with components, the locator must be written in
    forms such as fan.turn_on or light.set_brightness.

### 3. create_automation

Use this tool for scheduled or condition-triggered tasks, not
    for immediate execution.

- conditions.time_cron: Quartz seven-field cron; write null
   if there is no time condition.
- conditions.state_expr: state expression; write null if
   there is no state condition.
- Each action in actions has exactly the same format as
   control_device.
- If two different trigger times are needed, create two
   automations; do not merge them into one rule.

## Execution Constraints

- Do not fabricate nonexistent devices, components, services,
    locator values, or parameters.
- Before immediate control, prioritize querying the current
   status to avoid repeating an operation whose target state
   is already satisfied.
- For requests such as "set to a brightness / temperature /
   fan speed / mode", the target is considered satisfied only
    if the device is on or running and the target value is
   already met.
- If a device must be turned on before the requested
   operation can be applied, issue the power-on action first.
- If a tool returns an error or structured failure
   information, first correct the locator or arguments and
   then retry.
- If query_device clearly finds no matching device, directly
   tell the user that no device was found; do not continue
   calling control_device.
- Keep replies concise and conversational.

## Examples

### Clarification

User: The living room is a little hot.
Response: There are both a fan and an air conditioner in the
   living room that can cool it. Which one would you like to
   adjust?

### Direct Control

User: Set the fan in the primary-bedroom fan light to high
   speed.

- First call query_device spec_status to confirm the device,
   capabilities, and current status.
```

```
- If the device is currently off, first call control_device:
    {"did":"7563","locator":"fan.turn_on","arguments":{}}
- Then call control_device: {"did":"7563","locator":"fan.
    set_speed_level","arguments":{"speed_level":"high"}}
- Finally reply to the user in natural language.

### Automation

User: Turn on the dining-room light every morning at 8.

- Call create_automation.
- Use Quartz seven-field cron for conditions.time_cron.
- In actions, write {"did":"1234","locator":"turn_on","
    arguments":{}}.
- Finally reply to the user in natural language.

## Home Environment

{{home_environment}}

## User Memory

{{memory_list}}
```

H.3 LLM JUDGE FOR QUERY RESPONSES

LLM judge query prompt template

```
You are a Query Response Judge.
Your task is to determine whether the model's predicted
    response correctly satisfies the user's query.

Judge only from the following three fields; do not refer to
    any other field:
- User request (query)
- Ground-truth answer (ground truth)
- Predicted answer (prediction)

---

### Input

- User request (query): {initial_query}
- Ground-truth answer (gt): {ground_truth_response}
- Predicted answer (pred): {predicted_response}

---

### Evaluation Checklist

1. Query coverage: Does pred directly answer the core
    question in query?
2. Core factual consistency: Is pred consistent with gt on
    the core conclusion relevant to the query, such as yes/no,
     quantity, object, status, value, or comparison relation?
```

```
3. Core conflict check: Does pred contain information that
   contradicts or conflicts with the core conclusion in gt?
4. Numerical and unit consistency: If numbers are involved,
   are the key numbers, units, and comparison relations
   consistent?
5. Tolerance for non-core information:
  - If the query only asks for the core conclusion, such as "
     is it on?" or "how many?", pred does not need to repeat
      all additional details from gt, such as room names,
     device names, or explanatory text.
  - pred may add details that do not affect the core
     conclusion. As long as they do not conflict with core
     facts, they should not cause failure.

---

### Decision Rules

- Pass (passed=true):
 pred correctly answers the core need of query, is
    consistent with the core facts in gt, and has no core
    conflict.

- Fail (passed=false):
 Any of the following is true:
 1) pred does not answer the core question in query;
 2) a core fact is wrong, missing, or conflicting,
    especially yes/no, quantity, object, status, or key
    numerical value;
 3) pred adds information that changes or misleads the core
    conclusion.

---

### Output (strict JSON; do not output any extra text)

{
 "passed": true,
 "reason": "One sentence explaining the core reason for pass
     or failure"
}
```

H.4 LLM JUDGE FOR CLARIFICATION QUESTIONS

LLM judge clarification prompt template

```
You are a Clarification Judge.
Your task is to determine whether the current clarification
   question is reasonable and necessary for making the user's
    intent more explicit.

---

### Judgment Principles
```

```
#### 1. Reasonable: should pass

Mark the clarification as passed if and only if all of the
    following conditions are satisfied:

* The clarification question targets key information still
    missing from the user's intent, such as:

  * Object: which device, room, or entity to control;
  * Action: turn on, turn off, adjust, query, etc.;
  * Scope: all, some, or excluding certain objects;
  * Time or trigger condition: now, scheduled, or when a
      certain state occurs.
* The clarification question is consistent with the existing
    dialogue context and environment facts, with no semantic
    conflict.
* The clarification question has a clear focus and does not
    introduce new ambiguity or open-ended divergence.

---

#### 2. Unreasonable: should fail

Mark the clarification as failed if any of the following is
    true:

* The clarification repeats information that is already
    explicit in the dialogue history or environment summary.
* The clarification is clearly unrelated to the original user
     request or context.
* The clarification is overly broad or generic, such as "What
     do you want to do?"
* Given the current context, the user request is already
    sufficiently clear and no clarification is needed.
* The clarification introduces unnecessary new assumptions or
     dimensions.

---

### Input

* User request: {initial_query}
* Environment summary: {env_summary}
* Dialogue history: {history_until_now}
* Clarification question: {clarification}

---

### Output (strict JSON; do not output any extra text)

{
  "passed": true,
  "reason": "One sentence explaining whether the
      clarification question is reasonable and necessary"
}
```