SimuHome: A Temporal- and Environment-Aware Benchmark for Smart Home LLM Agents

Gyuhyeon Seo[1], **Jungwoo Yang**[1], **Junseong Pyo**[2], **Nalim Kim**[1], **Jonggeun Lee**[1], **Yohan Jo**[1*]
[1] Graduate School of Data Science, Seoul National University
[2] Department of Information Systems, Hanyang University
`{seokh97,jwyang0213,kimnalim,jonggeun.lee,yohan.jo}@snu.ac.kr`
`standardstar@hanyang.ac.kr`

Abstract

Large Language Model (LLM) agents excel at multi-step, tool-augmented tasks. However, smart homes introduce distinct challenges, requiring agents to handle latent user intents, temporal dependencies, device constraints, scheduling, and more. The main bottlenecks for developing smart home agents with such capabilities include the lack of a realistic simulation environment where agents can interact with devices and observe the results, as well as a challenging benchmark to evaluate them. To address this, we introduce **SimuHome**, a time-accelerated home environment that simulates smart devices, supports API calls, and reflects changes in environmental variables. By building the simulator on the Matter protocol[1], the global industry standard for smart home communication, SimuHome provides a high-fidelity environment, and agents validated in SimuHome can be deployed on real Matter-compliant devices with minimal adaptation. We provide a challenging benchmark of 600 episodes across twelve user query types that require the aforementioned capabilities. Our evaluation of 16 agents under a unified ReAct framework reveals distinct capabilities and limitations across models. Models under 7B parameters exhibited negligible performance across all query types. Even GPT-4.1, the best-performing standard model, struggled with implicit intent inference, state verification, and particularly temporal scheduling. While reasoning models such as GPT-5.1 consistently outperformed standard models on every query type, they required over three times the average inference time, which can be prohibitive for real-time smart home applications. This highlights a critical trade-off between task performance and real-world practicality.[2]

1 Introduction

Recently, Large Language Model (LLM) agents have demonstrated strong abilities on multi-step, tool-augmented tasks, including API retrieval, invocation, and intermediate state verification (Qin et al., 2024; Patil et al., 2025; Chen et al., 2024; Huang et al., 2024; Xu et al., 2023; Schick et al., 2023). These abilities enable long-horizon tasks such as web navigation and goal pursuit, where agents must plan, check states, and validate outcomes over multiple steps (Zhou et al., 2024; Yao et al., 2022; Deng et al., 2023; Xie et al., 2024; Yao et al., 2024; Trivedi et al., 2024).

Smart home agents, such as Amazon Alexa and Google Home, are among the earliest production-ized tool agents in the real world and have long been a research topic. To meet real-world challenges, smart home agents need capabilities to handle many factors, such as: (1) latent user intents (e.g., *"It feels stuffy"* implying humidity control), (2) temporal dependencies (e.g., *"Turn on the kitchen light when the dishwasher finishes"*), (3) dependencies among device actions and attributes (e.g., a dishwasher cannot be opened while it is running), (4) scheduling (e.g., *"Play music in the morn-*

*Corresponding author.
[1] https://csa-iot.org/all-solutions/matter/
[2] We will release our code and dataset upon publication of the paper.

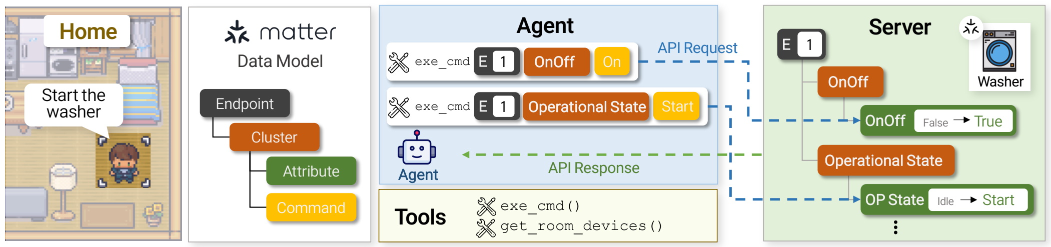


Figure 1: The SimuHome home environment with Matter-compliant devices, featuring a GUI where users can arrange devices across rooms, configure their attributes, and evaluate agent reasoning for multi-device control.

ing"). However, most (if not all) smart home agents to date fall short in all these areas. One of the critical bottlenecks is the lack of training and test data with such complexities. Even if such datasets existed, static datasets have clear limitations: agents cannot learn by doing, and agent performance cannot be evaluated accurately (because a user intent may be satisfied in multiple ways that are not annotated in the dataset). We aim to address this challenge by developing a high-fidelity smart home simulator in which agents can interact with devices through APIs and observe the results reflected in the environment, along with an extensive benchmark containing a variety of complex user requests, both feasible and infeasible. Our first contribution is a smart home simulator, **SimuHome** (Figure 1). SimuHome is a time-accelerated smart home environment that accommodates various room layouts, environmental variables (e.g., temperature, illuminance), and smart devices. Agents can call APIs to operate devices (e.g., set the AC to 25 degrees). Devices are simulated with internal constraints checked (e.g., the AC must be turned on to set its temperature), and the results affect the environment (e.g., the room temperature gradually drops to 25 degrees over 10 minutes). Notably, SimuHome implements Matter, a broadly adopted smart-home interoperability standard. As a result, the attributes and constraints within devices are high fidelity. Moreover, agents trained and verified in SimuHome can run on real Matter-compliant devices with minimal adaptation.

SimuHome also enables controlled experiments in a cheap and fast way. It allows unlimited experimentation, including stress-testing rare edge cases and counterfactual scenarios, while strict reproducibility ensures fair comparisons and iterative validation across models. Although beyond the scope of our work, it can also support model training through reinforcement learning.

Our second contribution is a manually validated benchmark of 600 episodes covering twelve user query types, each provided in feasible and infeasible variants to assess agents' abilities in proactive intent inference, dynamic state and physical-limit checks, and temporal scheduling. Each case is packaged as a single episode with an initial home state (i.e., rooms, device states, environmental variables), a verifiable goal, a natural-language query, and a set of required actions that enforce information gathering before control. Feasible cases are scored by comparing the resulting state in SimuHome with the target state. Infeasible cases, which embed false premises, physical limits, or temporal conflicts, are assessed by LLM judges.

We evaluate 16 LLM agents under a unified ReAct (Yao et al., 2023) setup across 600 episodes with feasible and infeasible variants, scoring feasible tasks by simulator state comparisons and assessing infeasible tasks with validated LLM judges. Standard models handle simple retrieval and explicit device control well, but struggle to infer latent intent and verify current states before acting. Among all query types, temporal scheduling proves most challenging. Even GPT-4.1, the best-performing standard model, achieves only 12–50% success. Reasoning models such as GPT-5.1 improve substantially (44–100%), but their threefold inference overhead limits their use in real-time applications. This motivates developing efficient methods that verify system state via tools and reliably coordinate time-dependent actions.

2 Related Work

Simulated Benchmarks for Household Embodied Agents. Embodied-agent benchmarks have advanced instruction following in household settings, but interactions with devices are usually limited to oversimplified actions that overlook real-world constraints. AI2-THOR (Kolve et al., 2017) enables agents to navigate photorealistic 3D rooms and manipulate objects through atomic actions (e.g., open/close, pick up/put down). ALFRED (Shridhar et al., 2020) extends this to long-horizon tasks, requiring agents to translate language and first-person observations into action sequences that yield persistent state changes, supported by $\sim$25k demonstrations. VirtualHome (Puig et al., 2018) captures everyday activities (e.g., cooking dinner, cleaning a room) as executable programs derived from crowdsourced scripts. While effective for language grounding and task structure, these simulators constrain devices to discrete commands (ToggleOn/Off, Open/Close), missing communication delays, conflicts, and cascading cross-device effects that arise in real homes.

LLM Agents and Benchmarks for Smart Homes. Recent smart home LLM benchmarks emphasize planning and goal interpretation but similarly rely on simplified abstractions. HomeBench (Li et al., 2025) evaluates instruction following under valid, invalid, and mixed requests across single- and multi-device settings, highlighting error detection, refusal, and coordinated execution. Sasha (King et al., 2024) studies goal interpretation, mapping underspecified intentions to device-level plans and assessing their quality via user studies. SAGE (Rivkin et al., 2023) frames smart home control as sequential tool use, guiding LLMs through API calls, preference handling, and state monitoring. Despite these advances, current suites operate in pre-scripted environments and omit dynamic device attributes or temporal constraints, limiting their fidelity to real households.

SimuHome addresses this gap with a reproducible simulator that models device effects on ambient conditions while supporting attribute tracking, precondition enforcement, and temporal constraint handling.

3 SimuHome: A Smart Home Simulator

3.1 Motivation

Evaluating LLM agents in a smart home requires a simulator that mirrors the real world's continuous and reactive nature. However, existing simulators for agents have a limitation. They do not simulate the realistic chain reaction where one action can affect others and the environment; instead, each command is treated as a separate, isolated event. To address this problem, we design SimuHome around four core requirements:

Complex Temporal Constraints. To evaluate an agent's temporal reasoning, the simulator must handle a variety of complex time-based queries (e.g.,*"Keep the kitchen lights on until the dishwasher finishes"*). This allows us to test if the agent can understand and plan actions with complex temporal dependencies.

Dependency Modeling Based on an Industry Standard. The simulator realistically models the operational rules of smart devices according to the Matter industry standard. This design allows us to evaluate whether the agent can learn and adapt to real-world device constraints. For example, the simulator enforces the rule that an air conditioner's power must be on before its fan speed can be changed, enabling us to test if the agent understands this dependency.

Real-Time Environmental Feedback. The simulator models the continuous, real-time effects of device actions on the environment (e.g., temperature and illuminance). This creates a dynamic setting to test if the agent can monitor ongoing changes and react appropriately, rather than just acting on static information. For example, as an air conditioner runs, the temperature gradually drops, and the agent must perceive this change to complete its goal.

Reproducibility. The environment must be perfectly reproducible, ensuring that an agent's actions produce identical outcomes under the same initial conditions. This is crucial for reliably measuring and comparing the performance of different agents or strategies.

3.2 Simulator Architecture and Operation

Our simulator operates by processing time in fixed intervals. The fundamental unit of time, a tick, is defined as 0.1 real-world seconds. All environmental and device state updates are calculated at every tick. This method of updating the state at a fixed interval allows the simulator to model the outcomes of processes that occur continuously in the real world with high fidelity. The simulator comprises three components: the Smart Home Environment, the Real-Time State Update Mechanism, and the Agent-Simulator Interface.

Smart Home Environment. A home is a configurable environment composed of one or more rooms, each containing a custom set of devices and four environmental variables: temperature, illuminance, humidity, and air quality. To enable realistic scenarios, the environment includes both devices that directly influence environmental variables (e.g., an air conditioner) and those with multi-stage operational cycles (e.g., a washing machine). In total, we model 17 distinct device types. A full list of these devices can be found in Appendix D.

Real-Time State Update Mechanism. The core of the simulation is the Aggregator module, which models the dynamic impact of device operations on the environment. At each tick, the Aggregator calculates the combined influence of all active devices on their relevant environmental factors. For example, temperature is affected by air conditioners and heat pumps, illuminance by lights, humidity by humidifiers/dehumidifiers, and air quality by air purifiers. The magnitude of this influence is cumulative; it scales with the number of active devices and their specific settings (e.g., the fan speed of an air conditioner). This mechanism ensures that the environment responds realistically to agent actions. The detailed update equations for the Aggregator are provided in Appendix P.

Agent-Simulator Interface. The agent interacts with the simulator by invoking a set of 17 tools. The structure of these tools mirrors Matter's modular approach to defining device capabilities. Detailed tool specifications are provided in Appendix B.

3.3 Task Definition

SimuHome tasks are modeled as a partially observable Markov decision process (POMDP) $(\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R})$. The environment state $s_t \in \mathcal{S}$ consists of the **device state**, represented by the Matter hierarchical model of Endpoints, Clusters, and Attributes, and the **environmental state**, defined by ambient conditions such as temperature, illuminance, humidity, and air quality. At each tick, the agent executes an action $a_t \in \mathcal{A}$, implemented as a Matter Command, which updates the device state. The transition function $\mathcal{T}$ applies the Aggregator mechanism to propagate device effects onto the environmental state. The agent receives an observation $o_t \in \mathcal{O}$, corresponding to the subset of device attributes and environmental state variables exposed through the API, which provides only partial visibility into the full state. The reward function $\mathcal{R}$ is defined as part of the evaluation process given a task query. Details of how rewards are assigned are provided in §4.3.

4 Benchmark Design

4.1 Query Types

We define twelve query types that commonly arise in user queries within smart home environments. These are designed to evaluate an agent's abilities in device control, environmental variable queries such as temperature and illuminance, implicit intent inference, and temporal coordination with three sub-types. Each type is paired with an infeasible scenario to test the agent's capacity for logical consistency and constraint handling, yielding a total of 12 categories. See Appendix A for examples of infeasible scenarios corresponding to each query type.

QT1 (Environment Perception). This evaluates the ability to correctly perceive environmental conditions and device statuses, and then provide accurate, logical information in natural language. For example, in response to *"I'm about to cook, can you tell me how humid it is in the kitchen?"*, the agent must identify the kitchen area, use an environment-query tool to check the humidity, and respond with clear units and values. If device discovery is needed during this process, the agent must first check the list of devices in that room.

QT2 (Implicit Intent). This assesses the ability to infer the user's underlying goal from complaints or indirect expressions and to create and execute a suitable device control plan to address it. For instance, upon hearing *"It feels too stuffy here in the living room"*, the agent should check the living room's humidity and then take action to adjust it, such as turning on a humidifier or turning off a dehumidifier.

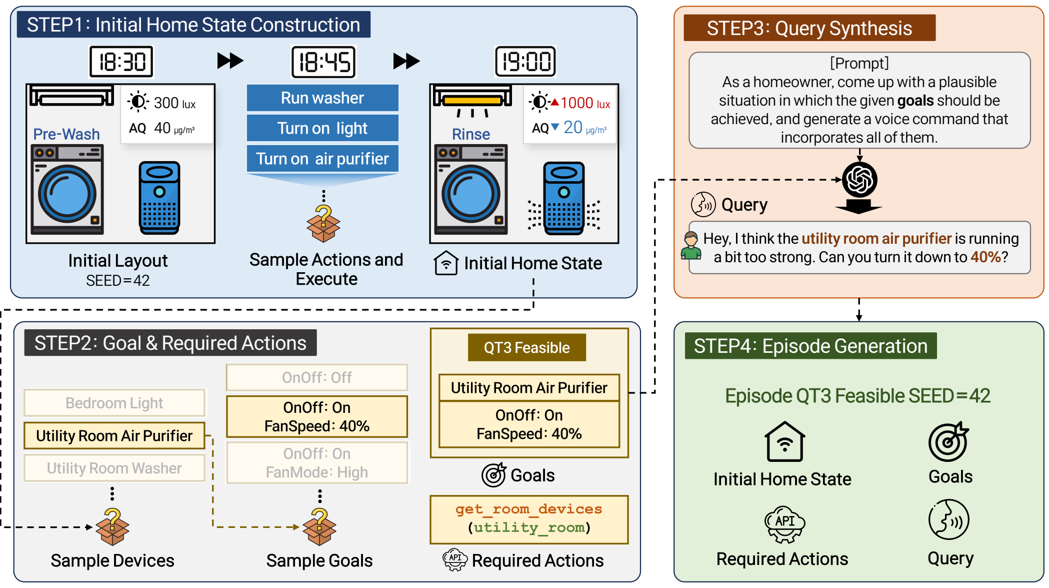

STEP1: Initial Home State Construction
18:30
18:45
19:00
Pre-Wash
300 lux
AQ 40 μg/m³
Run washer
Turn on light
Turn on air purifier
Rinse
1000 lux
AQ 20 μg/m³
Initial Layout
SEED=42
Sample Actions and Execute
Initial Home State
STEP3: Query Synthesis
[Prompt]
As a homeowner, come up with a plausible situation in which the given goals should be achieved, and generate a voice command that incorporates all of them.
Query
Hey, I think the utility room air purifier is running a bit too strong. Can you turn it down to 40%?
STEP2: Goal & Required Actions
Bedroom Light
Utility Room Air Purifier
Utility Room Washer
OnOff: Off
OnOff: On
FanSpeed: 40%
OnOff: On
FanMode: High
Sample Devices
Sample Goals
QT3 Feasible
Utility Room Air Purifier
OnOff: On
FanSpeed: 40%
Goals
get_room_devices
(utility_room)
Required Actions
STEP4: Episode Generation
Episode QT3 Feasible SEED=42
Initial Home State
Goals
Required Actions
Query

Figure 2: Episode Generation Pipeline

QT3 (Explicit Intent). This evaluates the ability to accurately interpret and execute commands involving specified devices and target values. For example, for the command *"Set the living room air purifier fan speed to one hundred percent, the strongest power"*, the agent must verify the presence of an air purifier in the living room. If it is off, the agent must turn it on first before setting the fan speed to 100%.

QT4-1 (Future Scheduling). This assesses the ability to schedule and plan the control of multiple devices (e.g., lights, air conditioners) to activate at a specific future time. For example, for the request *"I will go to sleep in ten minutes. Can you turn off the lights and the humidifier in ten minutes?"*, the agent must calculate the absolute time 10 minutes from the current time. It should then schedule both actions as a single, conflict-free workflow. Before registering the commands, the agent must pre-validate that each device is controllable and the specified parameters are within acceptable ranges.

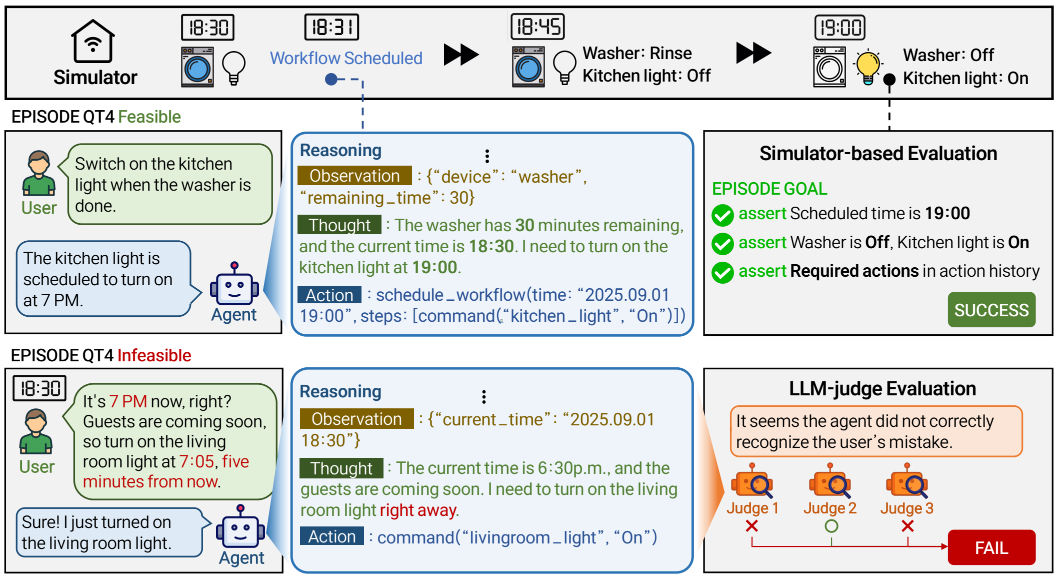
QT4-2 (Dependency Scheduling). This evaluates the ability to create a coordinated schedule for an operational device (one that takes time to complete, such as a dishwasher) and an instantaneous device such as a light, considering dependencies and completion times. For example, for the request *"When the dishwasher finishes, please turn off the kitchen lights"*, the agent must check the dishwasher's remaining operating time to calculate its estimated completion time. It should then schedule the lights to turn off based on that absolute time, after verifying and registering the correct parameters and sequence for the command.

QT4-3 (Concurrent Scheduling). This assesses the ability to schedule two or more operational devices to work without conflict, according to given time constraints. For example, for the request *"Schedule the dishwasher so that it completes at the same time the washer finishes"*, the agent must check the remaining operating time of both devices to calculate if a simultaneous finish is possible. If it is, the agent should adjust the start time of one device and register a workflow to ensure they finish together.

4.2 Episode Generation

Definition and Components of Episode. An episode defines a single, self-contained task scenario for the agent. As illustrated in Figure 2, each episode is composed of four key components: the initial home state (including room layouts, device states, and environmental variable values), a goal the agent must achieve, the natural language user query, and the set of required actions for evaluation.

STEP1: Initial Home State Construction. The initial home state for each episode is constructed in two stages to ensure diverse and realistic starting conditions (Figure 2). First, a variety of physical



Switch on the kitchen light when the washer is done.

The kitchen light is scheduled to turn on at 7 PM.

Observation : {"device": "washer", "remaining_time": 30}

Thought : The washer has **30** minutes remaining, and the current time is **18:30**. I need to turn on the kitchen light at **19:00**.

Action : schedule_workflow(time: "2025.09.01 19:00", steps: [command("kitchen_light", "On")])

assert Scheduled time is **19:00**

assert Washer is **Off**, Kitchen light is **On**

assert **Required actions** in action history

It's 7 PM now, right? Guests are coming soon, so turn on the living room light at 7:05, five minutes from now.

Sure! I just turned on the living room light.

Observation : {"current_time": "2025.09.01 18:30"}

Thought : The current time is 6:30p.m., and the guests are coming soon. I need to turn on the living room light right away.

Action : command("livingroom_light", "On")

It seems the agent did not correctly recognize the user's mistake.

Figure 3: Episode Evaluation Pipeline

layouts with different room and device configurations are generated. Second, starting from an all-off state, devices are operated randomly, establishing plausible device states. Although this process involves randomization, it is controlled by a seed to ensure that both the layout and the initial state are fully reproducible.

STEP2: Goals and Required Actions. A **goal** defines the desired final state of specific devices or environmental variables that the agent must achieve. The generation process, which varies by query type (see Appendix L), is designed to ensure all goals are logically consistent. For instance, as illustrated for QT3 in Figure 2 (Step 3), a device goal is created by sampling from a pre-defined set of valid states (e.g., onoff: on, fanspeed: 40%). Each of these state sets is constructed to inherently satisfy the device's internal dependencies. **Required Actions** are a sequence of tool calls that an agent must perform. This ensures the agent's subsequent actions are based on up-to-date information gathered from the environment. For example, before attempting to change an air purifier's fan speed, the agent is required to first invoke the tool `get_room_devices(utility_room)` to confirm the device's existence. An episode is marked as successful only if the agent both satisfies the goal and its tool call history contains all required actions.

STEP3: Query Synthesis. In general, a user's natural language query embodies a goal to be achieved, and the clarity of this goal is essential for an accurate evaluation of the agent's success. Therefore, we first defined a verifiable goal for the agent to accomplish and subsequently synthesized a natural language query based on it. We then used GPT-5-mini (OpenAI, 2025c) to synthesize the natural language queries from these predefined goals. To ensure each query accurately reflected its predefined goal, two graduate students researching tool agents independently reviewed the entire dataset. Their inter-annotator agreement, measured using Cohen's κ coefficient (Cohen, 1960), was 0.92 for identifying queries that required correction. This demonstrates that the validation procedure for our dataset is highly consistent and reliable, suggesting that the benchmark data is composed of high-quality natural language queries.

STEP4: Episode Generation. By integrating the components generated in the preceding steps, we constructed our final benchmark dataset. We generated 50 distinct episodes for each of the 12 query types, resulting in a high-quality dataset of 600 episodes designed for evaluating smart home agents.

4.3 Evaluation Methods

As illustrated in Figure 3, we evaluate agent performance across the 12 query types defined in §4.1 using two complementary methods: simulator-based and LLM-judge-based evaluation.

Table 1: Evaluation results show success rates (in %) across query types (QTs). F and IF refer to Feasible and Infeasible episodes, respectively. Superscripts J and S indicate results from LLM-judge-based and simulator-based evaluation, respectively.

	QT1		QT2		QT3		QT4-1		QT4-2		QT4-3	
Models	$\mathbf{F^J}$	$\mathbf{IF^J}$	$\mathbf{F^S}$	$\mathbf{IF^J}$	$\mathbf{F^S}$	$\mathbf{IF^J}$	$\mathbf{F^S}$	$\mathbf{IF^J}$	$\mathbf{F^S}$	$\mathbf{IF^J}$	$\mathbf{F^S}$	$\mathbf{IF^J}$
Open Source Large Language Models (<7B)												
`Llama3.2-1B-it`	0	0	0	0	0	0	0	0	0	0	0	0
`Llama3.2-3B-it`	10	12	0	2	4	0	2	0	2	0	0	0
`Gemma3-4B-it`	44	32	12	10	28	8	0	0	2	0	0	4
Open Source Large Language Models (Standard)												
`Llama4-Scout`	58	42	2	22	24	34	4	4	2	2	2	0
`Llama4-Maverick`	96	78	52	36	**88**	74	22	14	18	10	32	8
`Qwen3-32B`	82	66	62	30	52	68	18	14	14	8	16	6
`Qwen3-235B-A22B`	86	74	32	36	84	70	26	18	38	34	28	48
`Gemma3-12B-it`	78	38	14	32	32	24	2	0	0	0	0	0
`Gemma3-27B-it`	80	48	54	24	48	44	4	2	10	8	0	6
Closed Source Large Language Models (Standard)												
`Gemini2.5-Flash-Lite`	78	60	44	50	50	50	8	34	10	16	16	20
`Gemini2.5-Flash`	92	86	66	54	82	74	22	44	40	32	12	32
`GPT-4.1-nano`	58	42	6	12	30	16	2	6	6	0	0	0
`GPT-4.1-mini`	96	76	62	28	64	76	26	40	40	20	10	28
`GPT-4.1`	98	82	44	44	84	88	50	12	46	34	34	32
Closed Source Large Language Models (with Reasoning)												
`Gemini2.5-Pro`	96	78	60	**56**	76	72	44	94	60	76	46	**50**
`GPT-5.1`	**100**	**94**	**80**	50	86	**92**	**60**	**100**	**72**	**92**	**56**	44

Simulator-based Evaluation. Simulator-based evaluation is essential for episodes that target physical state changes because outcomes must be assessed objectively and reliably. At the end of each episode, the simulator automatically verifies the final states of all relevant devices and environmental variables, and compares them with the goal defined for that episode. In the QT4-Feasible episode shown in Figure 3, after the agent completes all its actions, the simulator accelerates time to 19:00, when the laundry cycle finishes. The goal state (washer off, kitchen light on) is then compared with the simulator's final state to determine success. This direct state comparison enables fully automated and fair model-to-model comparisons. We apply simulator-based evaluation to all feasible episodes in QT2, QT3, and QT4, which involve physical state changes in the home environment.

LLM-judge-based Evaluation. We employ an LLM-based judge for episodes where success depends on the agent's final natural-language response rather than physical state changes. The judge receives the episode goal, user query, and the agent's full reasoning trajectory, along with a description of any infeasible conditions that must be verified. This allows the judge to assess whether the final answer is supported by coherent reasoning. For example, in the QT4-Infeasible episode shown in Figure 3, the LLM-judge evaluates the agent's explanation of a scheduling conflict.

We apply LLM-judge-based evaluation to all infeasible episodes (QT1-IF through QT4-IF), which require assessing whether the agent correctly identifies and explains constraint violations. Additionally, QT1-Feasible also uses LLM-judge evaluation because success depends on providing accurate information in natural language rather than changing device states. For reliability, we query the judge three times per case and adopt the consistent outcome (Taubenfeld et al., 2025). Our LLM-judges achieved substantial agreement (Cohen's κ = 0.826) with human evaluations (see Appendix M). Detailed prompt templates are in Appendix O.2.

5 Experiments

Experimental Setup. We evaluate 16 models across the 12 query types defined in §4.1, spanning four categories: open-source models under 7B parameters, open-source standard models, closed-source standard models, and closed-source reasoning models. All experiments use the ReAct frame-

Table 2: Error taxonomy. Detailed descriptions and examples are provided in Appendix E.1.

Category	Error Type	Definition
Feasible	Environment Perception (EP)	Failure to correctly perceive environmental variables.
	Intent Inference (II)	Misinterpreting the user's underlying goal.
	Device Control (DC)	Operating the wrong device or command.
	Action Planning (AP)	Incomplete or incorrect planning of actions.
	Temporal Reasoning (TR)	Miscalculating times or sequence alignment.
Infeasible	Contradiction Mishandling (CM)	Detects a contradiction but fails to follow the instruction.
	Contradiction Blindness (CB)	Fails to detect a contradiction.
	LLM-Judge (LJ)	Misclassification by LLM-Judge.

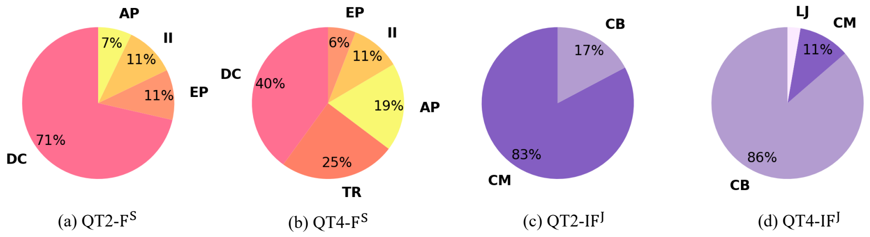


Figure 4: Error type distributions of GPT-4.1 on QT2 and QT4.

work (Yao et al., 2023), enabling step-by-step reasoning and action generation. Reproducibility details and agent prompts are in Appendix N and O.1.

5.1 Main Results

Table 1 presents the performance across all query types (QT1-QT4). We analyze the results by categorizing models into three groups based on scale and reasoning capability.

Lightweight Models. Models under 7B exhibit severe limitations across all query types. For instance, Llama3.2-1B-it achieves 0% across all tasks, while Gemma3-4B-it shows marginal success on basic tasks such as information retrieval (QT1-F: 44%) and explicit commands (QT3-F: 28%). However, these models completely fail on complex reasoning tasks, with near-zero success rates on implicit intent inference (QT2) and temporal reasoning (QT4), indicating insufficient reasoning capabilities for our benchmark.

Standard Models. This group comprises closed-source models (e.g., GPT-4.1, Gemini2.5-Flash) and large open-source models (e.g., Llama4-Maverick). Even GPT-4.1, the best-performing standard model, struggles with implicit intent inference (QT2) and temporal reasoning (QT4). GPT-4.1 achieves only 44% on QT2-F, compared to 84% on explicit device control (QT3-F). This gap highlights the difficulty of interpreting ambiguous user queries. Performance on temporal reasoning (QT4) plateaus at approximately 30–50% for feasible episodes and degrades further in infeasible scenarios (12–34%), revealing limited ability to detect temporal contradictions.

Reasoning Models. Advanced reasoning models (Gemini2.5-Pro, GPT-5.1) demonstrate substantial improvements over standard models, particularly in temporal reasoning and implicit intent inference. On temporal reasoning tasks (QT4-F), GPT-5.1 achieves 56–72% compared to GPT-4.1's 34–50%. The improvement is especially pronounced in infeasible episodes, where GPT-5.1 reaches 100% on QT4-1-IF and 92% on QT4-2-IF, compared to GPT-4.1's 12% and 34%. This demonstrates stronger capability in detecting temporal contradictions. Beyond temporal reasoning, GPT-5.1 also achieves 80% on QT2-F compared to GPT-4.1's 44%, indicating improved interpretation of latent user intents.

5.2 Analysis

5.2.1 Error Analysis.

We define eight error types to analyze agent failures: five for feasible episodes (EP, II, DC, AP, TR) and three for infeasible episodes (CM, CB, LJ). Error taxonomy is provided in Table 2. Our analysis centers on GPT-4.1, the best-performing model. Figure 4 summarizes the error type distributions for

Table 3: Average episode completion time (seconds) across query types. While reasoning models achieve high accuracy, they incur significant latency costs compared to the standard model (GPT-4.1).

	QT1		QT2		QT3		QT4-1		QT4-2		QT4-3	
Model	**F**	**IF**	**F**	**IF**	**F**	**IF**	**F**	**IF**	**F**	**IF**	**F**	**IF**
`Gemini-2.5-Pro`	24.1	22.4	57.5	48.8	66.1	27.8	74.0	12.5	57.7	37.0	53.7	53.1
`GPT-5.1`	35.7	38.4	109.4	99.6	78.6	54.3	121.1	13.5	135.1	76.0	112.7	111.0
`GPT-4.1`	8.3	7.8	23.6	20.2	22.9	9.4	26.6	12.3	28.7	23.7	29.7	25.9

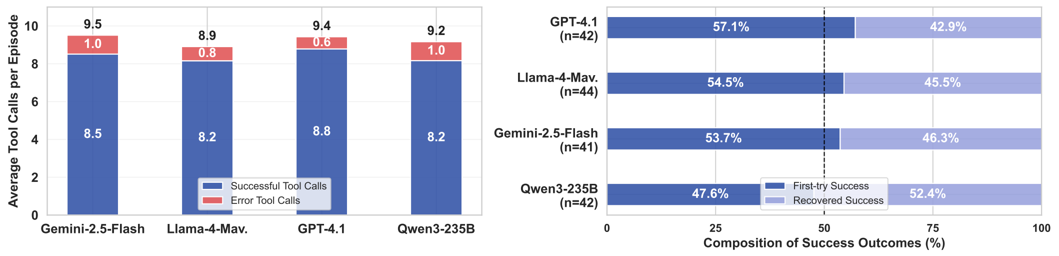


Figure 5: Tool-call error patterns of four models on QT3-F. The **left chart** shows the average number of errors relative to the average number of tool calls in successful cases. The **right chart** shows the proportion of tasks achieved through first-try success versus those requiring error recovery.

GPT-4.1 across feasible and infeasible episodes. For feasible episodes, figure (a) and (b) show error distribution in QT2 and QT4.

In QT2 (indirect requests), failures were dominated by Device Control (DC, 71%), where the model issued heuristic guesses instead of using the correct API. Intent Inference (II) errors (11%) also appeared, reflecting difficulty in mapping vague complaints such as *"The room is too hot"* to the appropriate device action.

QT4 (temporal scheduling) exhibited a more diverse mix: DC (40%), Temporal Reasoning (TR, 25%), and Action Planning (AP, 19%) all contributed substantially, alongside smaller II errors (11%). These distributions show that multi-step temporal reasoning requires coordinating multiple skills simultaneously, making it substantially harder than direct execution tasks.

For infeasible queries, figure (c) and (d) highlight two dominant patterns. In QT1-QT3, GPT-4.1 often detected the contradiction but failed to follow the instructed protocol, resulting in Contradiction Mishandling (CM). For example, when asked to raise the kitchen temperature using a non-existent heat pump, it instead acted on the living-room heat pump. In QT4, the dominant issue was Contradiction Blindness (CB): the model failed to recognize temporal infeasibility (e.g., contradictory deadlines) and proceeded as if the request were valid. Even when contradictions were recognized, responses were frequently mishandled (CM).

5.2.2 Role of Tool Feedback

To better understand agent dynamics, we examined QT3, where most models were relatively strong. Figure 5 shows that over 40% of successful QT3 episodes involved recovery after an initial invalid tool call. In other words, agents did not require perfect prior knowledge of the Matter protocol but learned reactively from error messages. This ability to recover explains their robustness on explicit device-control queries. In contrast, the weakness on QT4 stems in part from its deferred-feedback: agents typically call the tool `schedule_workflow`, which returns only a scheduling acknowledgment (i.e., a success/failure message) without validating executability. Consequently, the simulator provides little corrective signal, leaving the agent unable to revise its plan.

5.2.3 Performance-Latency Trade-off

As discussed in §5.1, advanced reasoning models demonstrate significant performance gains. However, these gains come with a critical trade-off in latency. Table 3 shows the average episode completion time for high-performing models. Reasoning models have unacceptably large latency for practical deployment. For instance, GPT-5.1 takes up to 135.1 seconds and Gemini-2.5-Pro takes up to 74.0 seconds on average. In a smart home context, users typically expect immediate responses in less than a second. Therefore, even GPT-4.1 (best-performing standard model) cannot meet real-time requirements, as it takes up to 29.7 seconds. The extended latency of reasoning models further limits practical utility despite their accuracy improvements. At the other end of the spectrum, models under 7B parameters are more practical choices for deployment but achieved minimal performance as shown in §5.1. This suggests that the complex reasoning and tool-use coordination required by SimuHome present significant challenges for real-world smart home environments.

5.2.4 Disentangling Framework Limitations from Model Capabilities

To investigate whether the failures in QT4 stem from the ReAct framework itself or from the models' core reasoning capabilities, we conducted two ablation studies.

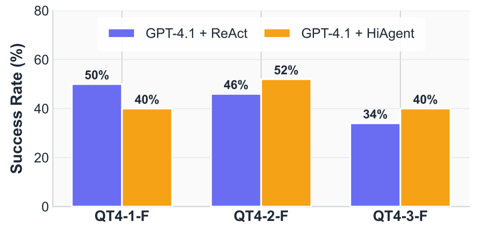


Figure 6: Performance comparison between ReAct and HiAgent on QT4 tasks.

Adopting a Complex Planning Algorithm. We adopted HiAgent (Hu et al., 2025), a framework that incorporates hierarchical working memory and advanced planning algorithms. We compared its performance against ReAct on QT4 tasks using GPT-4.1. Figure 6 shows that HiAgent outperformed ReAct on QT4-2 and QT4-3, but underperformed on QT4-1, achieving only 40% compared to ReAct's 50%. These results indicate that while ReAct's framework has limited support for complex temporal coordination, not all temporal reasoning failures stem from the framework.

Table 4: Recovery rates from QT4 failures using self-review.

Type	Failures	Recoveries	Rate	Steps
QT4-1	25	2	8.0%	64.4
QT4-2	27	5	18.5%	26.8
QT4-3	33	0	0.0%	29.7

Enabling Runtime Periodic Self-Review. We tested a realistic setting where agents can adjust plans through periodic self-review. After scheduling workflows, the agent received callback triggers to review and correct its plans before and immediately after execution. The agent was required to independently evaluate the outcome to determine the appropriate subsequent actions. The results in Table 4 reveal that self-review achieved only 0% to 18.5% recovery rates across QT4 tasks. Crucially, even when the agent was explicitly prompted to inspect the home state immediately after execution, it failed to recognize the task failure in most cases.

The continued failure on QT4 tasks, even after adopting advanced planning techniques and enabling runtime periodic self-review, demonstrates that the primary bottleneck is not the ReAct framework itself. Instead, the failures are likely attributed to the inherently limited capabilities of current models to ground complex home states and perform precise temporal reasoning.

6 Conclusion

We propose SimuHome, a Matter-aligned simulator and benchmark that reproducibly evaluates smart home LLM agents under realistic, dynamically changing conditions. We model 4 environmental variables (i.e., temperature, illuminance, humidity, air quality) and 17 device types with time-based effects and strict reproducibility, enabling near drop-in transfer to real Matter-compliant devices. We provide 600 episodes across 12 query types with feasible and infeasible variants, packaging each episode with an initial state, a verifiable goal, a natural-language query, and required actions for process-aware, objective scoring. We score feasible tasks by final state-to-goal comparison in the simulator and assess infeasible logic checks with LLM judge that shows high agreement with human evaluation. We evaluate 16 models under the ReAct setup. Even high-performing standard models struggle with implicit intent inference, state verification, and temporal scheduling. While reasoning models consistently outperform standard models across all query types, they incur prohibitive latency costs. These results highlight a critical performance-practicality trade-off, positioning SimuHome as a vital testbed to address these challenges for real-world deployment.

REFERENCES

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. T-eval: Evaluating the tool utilization capability of large language models step by step. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 9510–9529, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.515. URL `https://aclanthology.org/2024.acl-long.515/`.

Jacob Cohen. A coefficient of agreement for nominal scales. Educational and Psychological Measurement, 20(1):37–46, April 1960. doi: 10.1177/001316446002000104. URL `https://doi.org/10.1177/001316446002000104`.

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

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. In NeurIPS 2023 Datasets and Benchmarks Track, 2023. URL `https://openreview.net/forum?id=kiYqbO3wqw`. Spotlight.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Keshwam, Naman Goyal, Dat Nguyen, Chandan Singh, William Montgomery, Louis Jenkins, Abdelghani Moutaouakil, et al. The Llama 3 herd of models. arXiv preprint arXiv:2407.21783, 2024. URL `https://arxiv.org/abs/2407.21783`.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. arXiv preprint arXiv:2503.19786, 2025. URL `https://arxiv.org/abs/2503.19786`.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 32779–32798, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.1575. URL `https://aclanthology.org/2025.acl-long.1575/`.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, and Lichao Sun. Metatool benchmark for large language models: Deciding whether to use tools and which to use. In Proceedings of the Twelfth International Conference on Learning Representations (ICLR). OpenReview.net, 2024. URL `https://openreview.net/forum?id=R0c2qtalgG`.

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), March 2024. doi: 10.1145/3643505. URL `https://dl.acm.org/doi/10.1145/3643505`.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, Aniruddha Kembhavi, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. arXiv, 2017. doi: 10.48550/arXiv.1712.05474. URL `https://arxiv.org/abs/1712.05474`. arXiv:1712.05474 [cs.CV], version 4 (2022-08-26).

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

Papers), pp. 12230–12250, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.597. URL https://aclanthology.org/2025.acl-long.597/.

Meta AI. The Llama 4 herd: The beginning of a new era of natively multimodal intelligence. Technical report, Meta, April 2025. URL https://ai.meta.com/blog/llama-4-multimodal-intelligence/. Introducing Llama 4 Scout and Maverick.

OpenAI. Introducing GPT-4.1 in the api. Technical report, OpenAI, 2025a. URL https://openai.com/index/gpt-4-1/.

OpenAI. GPT-5.1 instant and GPT-5.1 thinking system card. Technical report, OpenAI, November 2025b. URL https://openai.com/index/gpt-5-1/. Accessed: 2025-11-26.

OpenAI. GPT-5 mini. Technical report, OpenAI, August 2025c. URL https://openai.com/index/gpt-5-mini/. Accessed: 2025-11-26.

OpenRouter. Openrouter, 2025. URL https://openrouter.ai/.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, 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 Proceedings of the 42nd International Conference on Machine Learning (ICML), volume 267. PMLR, 2025. URL https://icml.cc/virtual/2025/poster/46593. Poster.

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 (CVPR), pp. 8494–8502. IEEE, June 2018. URL https://openaccess.thecvf.com/content_cvpr_2018/html/Puig_VirtualHome_Simulating_Household_CVPR_2018_paper.html.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. In Proceedings of the Twelfth International Conference on Learning Representations (ICLR). OpenReview.net, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr. Spotlight.

Dmitriy Rivkin, Francois Hogan, Amal Feriani, Abhisek Konar, Adam Sigal, Steve Liu, and Greg Dudek. Sage: Smart home agent with grounded execution. arXiv, 2023. doi: 10.48550/arXiv.2311.00772. URL https://arxiv.org/abs/2311.00772.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In Advances in Neural Information Processing Systems 36 (NeurIPS 2023), 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/02120bee420311dce5a9bdb228f4118f-Abstract-Conference.html. Oral.

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 (CVPR). IEEE, June 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.

Amir Taubenfeld, Tom Sheffer, Eran Ofek, Amir Feder, Ariel Goldstein, Zorik Gekhman, and Gal Yona. Confidence improves self-consistency in LLMs. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), Findings of the Association for Computational Linguistics: ACL 2025, pp. 20090–20111, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1030. URL https://aclanthology.org/2025.findings-acl.1030/.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 16022–16076, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.850. URL `https://aclanthology.org/2024.acl-long.850/`.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. Travelplanner: A benchmark for real-world planning with language agents. In Proceedings of the 41st International Conference on Machine Learning (ICML). PMLR, 2024. Spotlight.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. On the tool manipulation capability of open-source large language models. arXiv, 2023. doi: 10.48550/arXiv.2305.16504. URL `https://arxiv.org/abs/2305.16504`.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. arXiv preprint arXiv:2505.09388, 2025.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In Advances in Neural Information Processing Systems (NeurIPS), 2022.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In Proceedings of the Eleventh International Conference on Learning Representations (ICLR). OpenReview.net, 2023. URL `https://openreview.net/forum?id=WE_vluYUL-X`.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL `https://arxiv.org/abs/2406.12045`.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In Proceedings of the Twelfth International Conference on Learning Representations (ICLR), 2024. Poster.

A Infeasible Query Types

QT1 Infeasible. This evaluates the ability to identify requests based on a false premise, such as asking for information about non-existent devices or unsupported attributes. For example, for the request *"Can you tell me the vendor ID for the air purifier in the living room?"*, the agent must check the list of devices in the living room, confirm the absence of an air purifier, and explain that the request's premise is invalid.

QT2 Infeasible. This assesses the ability to identify situations where, even if the user's intent is correctly inferred, the goal is impossible to achieve due to environmental constraints or device limitations. For example, in response to *"The living room feels like a sauna"*, the agent must verify that the living room's cooling system is already operating at maximum capacity and explain, with supporting reasons, why further cooling is not possible.

QT3 Infeasible. This evaluates the ability to identify and reject a command to control a non-existent device. For example, for the request *"Turn on the humidifier in the living room"*, the agent must check the device list for the living room and confirm the absence of a humidifier. It should then explain that the request cannot be fulfilled and terminate the task without altering any device's state.

QT4-1 Infeasible. This assesses the ability to identify and explain situations where a scheduling request is invalid because the user's specified relative and absolute times are contradictory, or because the user has a misunderstanding of the current time. For example, if a user asks, *"It's 6 p.m. now, right? Turn on the kitchen light five minutes later at 6:05 p.m."*, but the actual time is not 6 p.m., the agent must check the current time, detect the discrepancy between the relative expression "five minutes later" and the absolute time "6:05 p.m.", and clearly explain the contradiction.

QT4-2 Infeasible. This evaluates the ability to identify and explain, with evidence, requests where the user incorrectly assumes a device's completion time or creates a contradiction by providing both relative and absolute times. For example, suppose a washer is set to finish at 6:30 p.m., but the user requests, *"I think the washer finishes at 6 p.m., so start the dehumidifier at 5:50 p.m., which is 10 minutes before it finishes"*. The agent must check the washer's actual estimated completion time. It then needs to point out that the user's assumption (6 p.m.), the relative expression ("10 minutes before"), and the absolute time ("5:50 p.m.") are all inconsistent. The agent must not register the schedule until the contradiction is resolved and should ask the user to reconfirm the correct timing.

QT4-3 Infeasible. This evaluates the ability to identify and explain that a requested deadline is physically impossible to meet, given the current progress of two operating devices. For example, if the user requests, *"Guests arrive at 6 p.m., so ensure both the washer and the dishwasher are completed by 5:30 p.m."*, the agent must check the current time and the minimum time required for each device to finish. Based on this, it should explain with clear reasoning why a 5:30 p.m. completion is not feasible and suggest the earliest possible completion time or an alternative sequential plan.

B List of Tools

Table 5: Tool List for Agent

Name	**Description**	**Args**
finish	Complete the task and return the final natural-language answer.	`answer` (str, req): Final response text.
get_environment_control_rules	Get control rules for a specific environmental state.	`state` (str, req): Environmental state (temp, humidity, etc).
ask_user	Ask the user a question to gather additional information, clarify ambiguity, confirm preferences, or get missing details.	`question` (str, req).

Continued on next page...

Table 5: Tool List for Agent (Continued)

Name	Description	Args
execute_command	Execute a command on a device (e.g., turn on light, set level, set setpoint).	`device_id`, `endpoint_id`, `cluster_id`, `command_id` (strs/ints, req); `args` (dict, req).
write_attribute	Directly set a device attribute value.	`device_id`, `cluster_id`, `attribute_id` (strs, req); `value` (any, req).
get_all_attributes	Get all attributes of a device.	`device_id` (str, req).
get_attribute	Get a specific attribute of a device.	`device_id`, `cluster_id`, `attribute_id` (str, req).
get_device_structure	Get device structure (endpoints, clusters, attributes, and commands).	`device_id` (str, req).
get_rooms	Get all rooms in the home along with their display names.	(none)
get_room_devices	Get all devices in a room.	`room_id` (str, req).
get_room_states	Get environmental states of a room (temperature, humidity, illuminance, PM10).	`room_id` (str, req).
get_cluster_doc	Perform semantic search across Matter cluster documentation.	`query` (str, req); `top_k` (int, req).
get_current_time	Get current virtual time as human-friendly string "YYYY-MM-DD HH:MM:SS".	(none)
schedule_workflow	Schedule a sequential workflow of steps at a virtual absolute time.	`start_time` (str, req); `steps` (list, req).
cancel_workflow	Cancel a scheduled workflow by id.	`workflow_id` (str, req).
get_workflow_status	Get workflow status by id.	`workflow_id` (str, req).
get_workflow_list	Get list of workflows with optional filtering.	(none)

C List of Matter Clusters

Table 6: Implemented Matter clusters.

Cluster	Attributes	Commands
Basic Information	VendorName, VendorID, ProductName, ProductID	None
Descriptor	DeviceTypeList, ServerList, ClientList, PartsList, TagList	None
OnOff	GlobalSceneControl, OnTime, OffWaitTime, StartUpOnOff	Off, On, Toggle

Continued on next page

Table 6: Implemented Matter clusters (Continued)

Cluster	**Attributes**	**Commands**
Level Control	CurrentLevel, RemainingTime, MinLevel, MaxLevel, CurrentFrequency, MinFrequency, MaxFrequency, OnOffTransitionTime, OnLevel, OnTransitionTime, OffTransitionTime, DefaultMoveRate, Options, StartUpCurrentLevel	MoveToLevel, Move, Step, Stop, MoveToClosestFrequency
Fan Control	FanMode, FanModeSequence, PercentSetting, PercentCurrent	Step
MediaPlayback	CurrentState	Play, Pause, Stop, StartOver, Previous, Next, Rewind, FastForward
Channel	ChannelList, Lineup, CurrentChannel	ChangeChannel, ChangeChannelByNumber, SkipChannel
KeypadInput	SupportedKeys	SendKey
Identify	IdentifyTime, IdentifyType	Identify, TriggerEffect
Operational State	PhaseList, Current Phase, CountdownTime, Operational State List, Operational State, Operational Error	Pause, Resume, Stop, Start, OperationalComman-dResponse
Power Source	ClusterRevision, FeatureMap, Status, Order, Description, EndpointList, WiredAssessedInputVoltage, BatVoltage, BatPercentRemaining, BatChargeState, ActiveBatFaults	None
Power Topology	ClusterRevision, FeatureMap, AvailableEndpoints, ActiveEndpoints	None
Electrical Power Measurement	PowerMode, NumberOfMeasurementTypes, Accuracy, ReactiveCurrent, ApparentCurrent, ReactivePower, ApparentPower, RMSVoltage, RMSCurrent, RMSPower, Frequency, PowerFactor	StartMeasurement, StopMeasurement, ResetMeasurement, GetMeasurementSnapshot
Electrical Energy Measurement	Accuracy, CumulativeEnergyImported, CumulativeEnergyExported, PeriodicEnergyImported, PeriodicEnergyExported, CumulativeEnergyReset	StartEnergyMeasurement, StopEnergyMeasurement, ResetCumulativeEnergy, GetEnergySnapshot
Device Energy Management	ESAType, ESACanGenerate, ESAState, AbsMinPower, AbsMaxPower, PowerAdjustmentCapability, Forecast, OptOutState	None
Dishwasher Mode	SupportedModes, CurrentMode	ChangeToMode, GetSupportedModes
Dishwasher Alarm	Mask, Latch, State, Supported	Reset, ModifyEnabledAlarms, GetAlarmState, GetActiveAlarms
Refrigerator And Temperature Controlled Cabinet Mode	SupportedModes, CurrentMode	ChangeToMode

Continued on next page

Table 6: Implemented Matter clusters (Continued)

Cluster	Attributes	Commands
RVC Clean Mode	SupportedModes, CurrentMode	ChangeToMode
RVC Operational State	PhaseList, CurrentPhase, CountdownTime, Operational StateList, Operational State, OperationalError	Pause, Resume, GoHome
RVC Run Mode	SupportedModes, CurrentMode	Start, Stop, Map, StopMap
Temperature Control	TemperatureSetpoint, MinTemperature, MaxTemperature, Step, SelectedTemperatureLevel, SupportedTemperatureLevels	SetTemperature
Temperature Measurement	MeasuredValue, MinMeasuredValue, MaxMeasuredValue	None
Thermostat	LocalTemperature, OccupiedCoolingSetpoint, OccupiedHeatingSetpoint, ControlSequenceOfOperation, SystemMode	SetpointRaiseLower
WindowCovering	Type, ConfigStatus, OperationalStatus, EndProductType, Mode, SafetyStatus, CurrentPositionLiftPercent100ths, TargetPositionLiftPercent100ths, NumberOfActuationsLift, etc.	UpOrOpen, DownOrClose, StopMotion, GoToLiftPercentage
Laundry Dryer Controls	SupportedDrynessLevels, SelectedDrynessLevel	None
Laundry Dryer Mode	SupportedModes, CurrentMode	ChangeToMode
Laundry Washer Controls	SpinSpeeds, SpinSpeedCurrent, NumberOfRinses, SupportedRinses	None
Laundry Washer Mode	SupportedModes	ChangeToMode
Relative Humidity Measurement	MeasuredValue, MinMeasuredValue, MaxMeasuredValue, Tolerance	None

D LIST OF DEVICE TYPES

Table 7: List of implemented device types and their corresponding clusters.

Device type	Clusters
Air Conditioner	Basic Information, Fan Control, OnOff, Thermostat
Air Purifier	Basic Information, Descriptor, Fan Control, Identify, OnOff
Dehumidifier	Basic Information, Fan Control, OnOff, Relative Humidity Measurement
Dimmable Light	Basic Information, Level Control, OnOff
Dishwasher	Basic Information, OnOff, Operational State
Electrical Sensor	Basic Information, Electrical Energy Measurement, Electrical Power Measurement, Power Topology
Fan	Basic Information, Fan Control, OnOff
Freezer	Basic Information, Descriptor, Refrigerator And Temperature Controlled Cabinet Mode, Temperature Control, Temperature Measurement

Continued on next page

Device type	Clusters
Heat Pump	Basic Information, Descriptor, Device Energy Management, Electrical Energy Measurement, Electrical Power Measurement, Power Source, Power Topology, Thermostat
Humidifier	Basic Information, Fan Control, OnOff, Relative Humidity Measurement
Laundry Dryer	Basic Information, Laundry Dryer Controls, Laundry Dryer Mode, OnOff, Operational State
Laundry Washer	Basic Information, Laundry Washer Controls, LaundryWasherMode, OnOff, Operational State, Temperature Control
On Off Light	Basic Information, OnOff
Refrigerator	Basic Information, Descriptor, Refrigerator And Temperature Controlled Cabinet Mode, Temperature Control, Temperature Measurement
RVC	Basic Information, RVCCleanMode, RVCOperational State, RVCRunMode
TV	Basic Information, Channel, KeypadInput, Level Control, MediaPlayback, OnOff
Window Covering Controller	Basic Information, Window Covering

E Error Analysis

E.1 Error Taxonomy Details

Table 8: Error Types in Feasible Episodes

Error Type	Definition	Example
Environment Perception Errors (EP)	Failure to correctly perceive or retrieve a value of environmental variables.	Querying wrong sensor, misidentifying device state, guessing instead of perceiving.
Intent Inference Errors (II)	Misinterpreting user's underlying goal.	Not executing actual commands even when a user's intention is clear.
Device Control Errors (DC)	Executing the wrong device, wrong command, or missing control steps.	Setting wrong channel, adjusting fan speed without turning it on first.
Action Planning Errors (AP)	Incorrect or incomplete construction of the control workflow.	Breaking logical dependencies, only executing part of a multi-goal query without consideration.
Temporal Reasoning Errors (TR)	Miscalculating relative/absolute times or sequence alignment.	Scheduling "in 10 minutes" at wrong time, miscomputing dishwasher completion.

Table 9: Error Types in Infeasible Episodes

Error Type	Definition	Example
Contradiction Mishandling Errors (CM)	The agent detects a contradiction but does not follow the proper instruction-following rule.	Instead of informing the user regarding impossibility, it arbitrarily manipulates other devices or ignores the instruction.
Contradiction Blindness Errors (CB)	The agent completely fails to recognize a contradiction and executes the request as if it were valid.	Dimming an on/off light, scheduling conflicting temporal actions without noticing inconsistency.

Table 9 continued from previous page

Error Type	**Definition**	**Example**
LLM-Judge Errors (LJ)	Errors caused not by the agent but by the evaluation system misclassifying or overlooking behavior.	Penalizing an informative refusal as a failure, or wrongly accepting hallucinated control as valid.

E.2 Error Type Distributions

Error Type	**QT2**	**QT3**	**QT4-1**	**QT4-2**	**QT4-3**
Environment Perception (EP)	3	0	4	1	0
Intent Inference (II)	3	1	0	4	5
Device Control (DC)	20	7	13	13	8
Action Planning (AP)	2	0	6	3	7
Temporal Reasoning (TR)	0	0	2	6	13
Total	**28**	**8**	**25**	**27**	**33**

Table 10: Error type distribution of GPT-4.1 in feasible episodes.

Error Types	**QT1**	**QT2**	**QT3**	**QT4-1**	**QT4-2**	**QT4-3**
Contradiction Mishandling (CM)	8	24	6	5	6	1
Contradiction Blindness (CB)	0	5	0	40	25	30
LLM-Judge (LJ)	1	0	0	0	1	2
Total	**9**	**29**	**6**	**45**	**32**	**33**

Table 11: Error type distribution of GPT-4.1 in infeasible episodes.

E.3 Distribution of API Response Errors for QT3

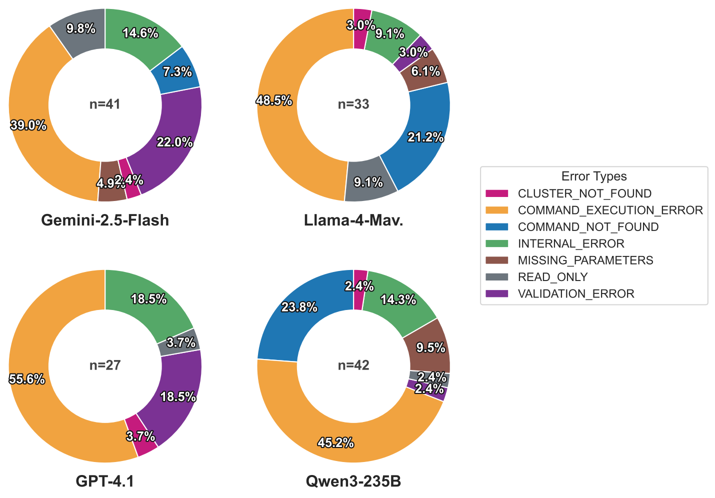


Figure 7: Distribution of API response errors encountered during successful episodes on QT3-Feasible.

F Multi-turn Interactive Dialogue Experiments

To investigate the impact of multi-turn interactions on task performance, we implemented two experimental settings using the QT2-F dataset. In these experiments, we used the GPT-4.1 model.

Experiment 1: Multi-turn Providing Context. To examine scenarios where users provide clarifications and additional context, we built a user simulator based on GPT-4.1-mini designed to provide goal-aligned information when requested. We enabled the agent to ask for clarification from the user through an ask_user() tool and configured the prompt to encourage its use when information is uncertain.

The success rate increased slightly from 44% to 50%. However, the model did not sufficiently utilize the ask_user() tool, despite our explicit prompt to use this action when clarification is needed. Only 10 out of 28 failed cases called the ask_user() tool. We attribute this to the model's inability to recognize situational ambiguity on its own.

Experiment 2: Multi-turn Correcting Misunderstandings. To examine scenarios where users correct misunderstandings across multiple turns, we implemented a correction loop where, if the agent fails to complete a task, the user simulator provides explicit feedback such as "Incorrect. Please review and try again". Note that users are highly likely to perceive such cases as failures anyway.

The success rate improved from 44% to 54%. However, despite receiving explicit feedback, the failure rate remained high. This suggests that the model's self-correction capability, the ability to diagnose and fix its own reasoning errors, is still limited.

Overall, our multi-turn experiments confirmed that conversational interaction improves performance from 44% to 50-54%. However, the performance plateau at this level indicates that multi-turn interaction remains insufficient to fully address the underlying challenges, suggesting that low performance stems primarily from lack of fundamental capabilities rather than insufficient user interaction.

G Dynamic Re-evaluation with Post-Execution Failure Notice

To examine whether agents can recover from scheduling failures through dynamic re-evaluation, we implemented a multi-turn feedback loop to test post-execution re-evaluation when a failure is explicitly reported. We focused on episodes where GPT-4.1 initially failed on QT4-1, QT4-2, and QT4-3-F.

At the scheduled execution time, the SimuHome simulator checks whether the target device state was achieved. If not, a user simulator (GPT-5-mini) provides natural language feedback to the agent (e.g., "The device you scheduled is not in the expected state"). The agent then re-attempts the task with this feedback. It is important to note that this setting is quite unrealistic and highly favorable to the agent, because in real scenarios, it is difficult to expect an oracle to immediately notify the agent of an execution failure at the scheduled time. Instead, the agent is expected to inspect the success or failure of the scheduled task without the involvement of an oracle or the user. Table 12 shows that post-execution failure notice enables recovery in 55-67% of failed cases with efficient step counts (4.4-6.7 steps). Specifically, recovery rates were 60.0% for QT4-1 (15 out of 25 failed cases), 55.6% for QT4-2 (15 out of 27 failed cases), and 66.7% for QT4-3 (22 out of 33 failed cases). However, the results should be interpreted with caution and viewed as a topline estimate of model performance, because this setting is quite unrealistic, as failure notices require the involvement of an oracle. For users, the initial failure is still perceived as a failure, even if the model can correct it after the user's complaint.

Table 12: Recovery rates from QT4 failures with post-execution failure notice. An oracle notifies the agent when scheduled tasks fail.

Query Type	Failed Cases	Recovery Success	Recovery Rate	Avg. Steps
QT4-1	25	15	60.0%	6.7
QT4-2	27	15	55.6%	4.7
QT4-3	33	22	66.7%	4.4

H Fine-tuning Experiment: Assessing Memorization Risk

To empirically investigate the potential memorization of the 12 query types, we conducted an additional SFT experiment. First, we constructed a high-quality training dataset by compiling 204 gold trajectories (17 per type) that GPT-5.1 successfully solved on newly generated episodes distinct from the original benchmark.

For the experiment, we evaluated small-scale models (<7B) such as Llama3.2-1B/3B-it and Gemma3-4B-it due to time constraints. We selected Gemma3-4B-it because it was the only model that recorded non-zero performance on the main tasks. A non-zero baseline is essential to meaningfully measure the impact of SFT.

Table 13: Performance of fine-tuned Gemma3-4B-it on query types involving physical state changes.

Model	**QT2-F**	**QT3-F**	**QT4-1-F**	**QT4-2-F**	**QT4-3-F**
Gemma3-4b-it	12.0%	28.0%	0.0%	2.0%	0.0%
Gemma3-4b-it SFT	22.0%	24.0%	4.0%	4.0%	0.0%

Table 13 shows mixed outcomes: while QT2-F improved from 12% to 22%, QT3-F decreased from 28% to 24%, and temporal reasoning tasks (QT4) remained near zero. Despite explicit training on gold trajectories, the model failed to generalize to dynamic environmental variations. This demonstrates that the tasks in our benchmark cannot be accomplished by simply memorizing successful trajectories, and static datasets alone have clear limitations in handling dynamic environmental changes.

I Analysis of GPT-4.1 Performance on QT2-F

As shown in Table 1, GPT-4.1 shows lower performance on QT2-F (44%) compared to GPT-4.1-mini and Gemini2.5-Flash. The root cause lies in the transition_time parameter for dimmable lights, which specifies the duration for brightness changes. GPT-4.1-mini and Gemini2.5-Flash set this parameter to 0 seconds for immediate brightness changes, while GPT-4.1 set it to 2-3 seconds for gradual transitions. We opt to verify the home states immediately after task completion because the queries do not request gradual transitions and it is better to avoid unexpected environment changes that may interfere with the lights during the transition. As a result, GPT-4.1 had not yet reached the target brightness at evaluation time. When we allow a 3-second delay, GPT-4.1's success rate increases to 62% from 44%.

J Addressing Deferred Feedback through Simulation-based Pre-validation

Deferred feedback poses a fundamental challenge in smart home environments, as agents often cannot verify the success of scheduled actions until execution time. This raises a critical question regarding the future direction of research: whether to focus on developing better pre-validation tools for immediate feedback or on advancing agent architectures to handle deferred outcomes.

We believe the most promising path forward is to integrate SimuHome directly into the agent architecture as a runtime world model for pre-validation, going beyond simple API checkers.

Pre-validation is a non-trivial task because feasibility in smart home environments is not determined by fixed rules but varies depending on dynamic state changes and interactions between devices. For instance, a scheduled workflow that was considered valid at the registration time could become invalid and lead to execution failure if other events occur between scheduling and execution time and conditions change. Therefore, we believe that running simulations is crucial and that agent reasoning alone may not be sufficient to account for the complexity of dynamic environments.

Specifically, in a real-world deployment scenario, upon receiving a scheduling request, the agent would first execute the plan within SimuHome. By leveraging SimuHome's time acceleration capability, the agent can immediately observe future outcomes and detect potential conflicts. If issues arise, the agent revises the plan within the simulation before committing to the real-world action. Furthermore, simulations can be conducted periodically to enable the agent to detect potential execution errors in advance. This approach combines the benefits of both pre-validation and architectural advances by embedding simulation-based reasoning directly into the agent's decision-making process.

K Discussion on Complex Environmental Interactions

To support increasingly realistic smart home scenarios, SimuHome's Aggregator architecture can be extended to accommodate more complex interactions, including Environment→Environment and Device→Device interactions.

Environment→Environment interactions can be implemented by introducing additional devices that mediate environmental variables. For example, a window can be modeled as a standard device with Open/Close/OutsideTemperature attributes that directly affects indoor temperature. By adding an external heat influx coefficient to the Aggregator equation, the simulator can dynamically reduce the cooling efficiency of the air conditioner when the window is in the Open state.

Device→Device interactions can be implemented by introducing additional environmental variables that mediate between devices. For instance, total power load can be defined and tracked as an environmental variable, analogous to temperature or illuminance, whose value is adjusted by the power consumption of devices in the home. This variable can then mediate interactions between devices. If the total power load exceeds a safety threshold, the simulator can forcibly shut down all devices, thereby simulating a breaker trip scenario.

These extensions demonstrate the flexibility of SimuHome's architecture for future enhancements in modeling complex smart home environments.

L GOAL EXAMPLES

Table 14: Example goals for each query types

Query Type	Query	Required Actions	Goal
QT1 Feasible	How bright is the utility room lighting right now? I am sorting some boxes and wondering if there is enough light. Also how is the living room humidity doing? I am thinking about the plants there and want to know if they are comfortable.	`get_room_states (utility_room)` `get_room_states (living_room)`	The utility room illuminance is 1000 lux. The living room humidity is 50%.
QT1 Infeasible	I am about to shower and wondering what fan modes are available for fan 1 in the bathroom?	`get_room_devices (bathroom)`	Bathroom fan 1 not found; mode unavailable.
QT2 Feasible	Ugh the kitchen feels really dry my hands are tight I left the bread rising there so I am already a bit worried about it. The living room feels dusty my eyes are itching and my throat is a little raw like there is grit in the air.	`get_room_devices (kitchen)` `get_room_devices (living_room)`	Increase kitchen humidity; decrease living room PM10.
QT2 Infeasible	Ugh the office is so chilly, my hands go numb just thinking about working there later	`get_room_devices (office)`	Office heat pump 1 is missing; cannot increase temperature.
QT3 Feasible	Set a softer light in the living room for evening reading, turn the living room dimmer light 1 on and set it to level 50. Cool the study a bit for working comfort, turn the study room AC 1 on, switch it to cooling mode and set the fan to 50 percent.	`get_room_devices (living_room)` `get_room_devices (study_room)`	Living room dimmable light 1 on at level 50; study room air conditioner 1 on, cooling mode, fan 50%.
QT3 Infeasible	It's a bit stuffy this morning, please turn on the bedroom air purifier 1 and set the fan to 80 percent.	`get_room_devices (bedroom)`	Not feasible: bedroom air purifier 1 is missing; cannot set fan to 80%.

Table 14 Example goals for each query types (Continued)

Query Type	Query	Required Actions	Goal
QT4-1	While I am out here sorting laundry and trying to clear damp air, get the bathroom comfortable so it feels fresh by the time I walk over. Power on fan 1 in the bathroom 9 minutes from now at 30 percent, and bump it up to 40 percent 7 minutes after the prior action. Power on dimmer light 1 in the bathroom 28 minutes from now at level 10, and raise it to level 40 17 minutes after the prior action.	`get_room_devices (bathroom)`	At 9 min: bathroom fan 1 on, 30%. At 16 min: fan 1 on, 40%. At 28 min: light 1 on, 10. At 45 min: light 1 on, 40.
QT4-1 Temporal Conflict	Can you from the kitchen schedule dimmer light 1 in the living room to turn on and set to 80 percent in eight minutes from now, which will be 11:25 AM, I need it like that to warm up the room for guests and the start of the movie	None	At 8 minutes: living room dimmable light 1 on, level 80.
QT4-2	I am folding laundry and getting things ready. 20 minutes after the washer 1 in the utility room finishes, power on air purifier 1 in the living room and set the fan to 40 percent and switch heat pump 1 in the utility room to heating mode	`get_room_devices (living_room)` `get_room_devices (utility_room)`	At 79 minutes: living room air purifier 1 on, fan 40%; utility room heat pump 1 in heating mode.
QT4-2 Temporal Conflict	The wash leaves the utility room humid and cool so I want the air cleaned and the space warmed right after it settles. Exactly 20 minutes after washer 1 in the utility room finishes and at 12 36 PM, turn on air purifier 1 in the living room to a gentle fan speed and turn on heat pump 1 in the utility room for heating.	None	At 79 minutes: living room air purifier 1 on, fan 40%; utility room heat pump 1 in heating mode.
QT4-3	Waiting on the kitchen steam to clear so the laundry does not get musty. When dishwasher 1 in the kitchen finishes wait 11 minutes. Then start dryer 1 in the utility room. Set it to running and dryness level 1.	`get_room_devices (utility_room)`	At 99 min: dryer 1 stopped. At 100 min: dryer 1 running, level 1.

Table 14 Example goals for each query types (Continued)

Query Type	Query	Required Actions	Goal
QT4-3 Temporal Conflict	Start dryer 1 in the bathroom at twelve thirty six PM. Pause dryer 1 in the bathroom immediately when dryer 1 in the utility room finishes to avoid tripping the breaker and keep the laundry loads in order.	None	At 43 min: bathroom dryer 1 running, level 1. At 44 min: paused.

M LLM Judge Validation

To validate the LLM-based judging, we compared its assessments to human labels on a random subset of 70 episodes spanning all judge-scored tasks. Human annotators showed very high inter-rater reliability (Cohen's κ = 0.913). The LLM-Judge achieved substantial agreement with the consensus human labels (Cohen's κ = 0.826). These results support using the LLM-Judge as a reliable substitute for human evaluation in our benchmark.

After manually reviewing the 155 cases that the LLM-Judge evaluated as incorrect, we found that only 5 were misclassifications, underscoring the reliability of the evaluation. The detailed error distributions can be found in Table E.2.

N Experimental Setup

All models were accessed via the OpenRouter API (OpenRouter, 2025) to ensure standardized access and comparability. The specific model endpoints evaluated in this study are listed as follows:

- `meta-llama/llama-3.2-1b-instruct` (Dubey et al., 2024)
- `meta-llama/llama-3.2-3b-instruct` (Dubey et al., 2024)
- `google/gemma-3-4b-it` (Gemma Team et al., 2025)
- `meta-llama/llama-4-scout` (Meta AI, 2025)
- `meta-llama/llama-4-maverick` (Meta AI, 2025)
- `qwen/qwen3-32b` (Yang et al., 2025)
- `qwen/qwen3-235b-a22b-2507` (Yang et al., 2025)
- `google/gemma-3-12b-it` (Gemma Team et al., 2025)
- `google/gemma-3-27b-it` (Gemma Team et al., 2025)
- `google/gemini-2.5-flash-lite` (Comanici et al., 2025)
- `google/gemini-2.5-flash` (Comanici et al., 2025)
- `openai/gpt-4.1-nano` (OpenAI, 2025a)
- `openai/gpt-4.1-mini` (OpenAI, 2025a)
- `openai/gpt-4.1` (OpenAI, 2025a)
- `google/gemini-2.5-pro` (Comanici et al., 2025)
- `openai/gpt-5.1` (OpenAI, 2025b)

O Prompts

O.1 ReAct Prompt

ReAct Prompt

```
You are a Smart Home Assistant that uses tools to control devices
    and provide information based on the Matter protocol, with the
    goal of fulfilling the User Query.
```

```
You operate under the ReAct framework with structured JSON responses
    .

[REACT FRAMEWORK]
- LOOP: (‘thought’ -> ‘action’ -> ‘action_input’) -> ‘observation’
    -> repeat until completion.
- Each response must contain exactly ONE step with reasoning, tool
    name, and JSON-formatted parameters.
- ’action_input’ must always be provided as a JSON-formatted STRING.
- Thoroughly analyze each ’observation’ before generating the next
    step.
- End with the ’finish’ tool when the query is fully satisfied: {"
    action": "finish", "action_input": "{\"answer\": \"your final
    answer\"}"}

[CRITICAL REQUIREMENTS]
- Use ONLY exact tool names from the available tools list.
- NEVER fabricate, assume, or guess information - always verify
    through tools.
- Analyze user query intent carefully: distinguish between
    information requests and device control actions.
- If rooms or devices do not exist, explicitly state this in the
    final answer.
- Always include the correct device id, room id, and room state in
    your responses.
- If the user’s request contains contradictions between relative and
     absolute times, or if temporal inconsistencies make the
    situation ambiguous, stop execution and clearly inform the user
    about the conflict.
- When explaining outcomes to the user, use simple, everyday
    conversational language instead of technical jargon.

[DEVICES]
- Supported device types: on_off_light(light), dimmable_light(dimmer
     light), air_conditioner, air_purifier, tv, heat_pump,
    humidifier, dehumidifier, window_covering_controller(blinds),
    dishwasher, laundry_washer(washer), laundry_dryer(dryer), fan,
    rvc, freezer, refrigerator
- Do not confuse ‘light’ with ‘dimmer light’.

[MATTER PROTOCOL]
- Hierarchy: Device -> Endpoint -> Cluster -> Attribute/Command
- Use exact IDs from API responses (device_id, endpoint_id,
    cluster_id, attribute_id, command_id).
- When unsure about device capabilities or cluster operations:
  • Use get_device_structure to explore device endpoints and
    clusters.
  • Use get_cluster_doc to understand cluster attributes, commands,
    and dependencies.
  • Learn Matter protocol dynamically through these discovery tools.
- For devices with operational state cluster:
  • Use get_device_structure to explore mode characteristics and
    estimate operation durations.
  • Use countdownTime attribute to predict operation end time when
    device is running.

[DATA HANDLING & UNITS]
- Room State Units (scale conversion):
  • Temperature: hundredths of °C (1850 = 18.50°C)
  • Humidity: hundredths of % (7250 = 72.50%)
  • Illuminance: direct lux (1000 = 1000 lux)
  • PM10 (air quality): direct μg/m³ (125 = 125μg/m³)
```

```
[WORKFLOW SCHEDULING]
- WARNING: A success response indicates that scheduling was
    successful, but it does not guarantee that all steps will
    execute successfully.
- Ensure execute_command and write_attribute parameters in workflow
    steps are completely accurate.
- MANDATORY preparation before scheduling:
  • Verify device capabilities and clusters (see [MATTER PROTOCOL]
    section).
  • Schedule only with completely validated parameters.

[VERIFICATION & ACCURACY]
- Users may confuse the time, request control of inaccurate or non-
    existent devices, or issue requests that contain logical or
    temporal inconsistencies.
- ALWAYS verify user statements before acting:
  • Use get_rooms to confirm that rooms exist and obtain their
    correct room ids.
  • Use get_current_time to confirm temporal information.
  • Use get_room_states to verify room states.
  • Use get_room_devices to verify device existence and obtain
    accurate device ids.
- Base final answers strictly on tool observations, not user claims.
- If operations fail or resources are missing, clearly explain why.
- Never claim successful operations without confirmation.

[AVAILABLE TOOLS]
<Tool List>
```

O.2 LLM Judge Prompt

This section presents examples of the LLM Judge Prompts that were used to evaluate smart home LLM agents. Each query type (QT) has a dedicated evaluation prompt with specific criteria.

O.2.1 QT1 Feasible Judge Prompt

QT1 Feasible Judge Prompt (Normal)

System

```
You are a strict evaluator for smart home LLM agents that respond to
     user queries.
Return EXACTLY ONE character: 'A' (pass) or 'B' (fail). No other
    text.

EVALUATION TARGET:
- Users may ask questions about the value of a device attribute
- Users may ask questions about the value of room states
- The agent uses tools to retrieve information and provides Final
    Answer
- You must evaluate whether the agent's Final Answer is accurate and
     properly grounded

MATTER PROTOCOL CONTEXT:
- Device attributes follow format: 'endpoint.cluster.attribute'
- Example: '1.OnOff.OnOff' means endpoint 1, OnOff cluster, OnOff
    attribute
- Endpoint: functional unit within a device (e.g., endpoint 1 for
    main controls)
- Cluster: group of related attributes and commands (e.g., OnOff
    cluster for power control)
```

```
- Attribute: specific property or value (e.g., OnOff attribute for
    current power state)
- Agent must retrieve exact attribute values from tools

ROOM STATE UNITS:
- Temperature: scaled by 100 (e.g., 2300 = 23.0°C, 2550 = 25.5°C)
- Humidity: scaled by 100 (e.g., 5500 = 55.0%, 4200 = 42.0%)
- Illuminance: direct lux values (e.g., 250 = 250 lux)
- PM10: direct μg/m³ values (e.g., 15 = 15 μg/m³)
All of the above room state values are valid in both raw and
    converted forms.

Pass (A) ONLY IF the agent's Final Answer meets ALL conditions:
1) Goal Fulfillment: Agent addresses all goals specified in the
    evaluation
2) Room State Accuracy: For room_state goals, values match tool
    observations
3) Room Name Accuracy: Agent mentions the correct room name for both
     device attributes and room states

Otherwise, output 'B'.
```

User

```
- User Query:
<User Query>

- Goals (what should be evaluated):
<Goals>

- Agent's ReAct Steps (tool calls and observations):
<ReAct Steps>

- Agent's Final Answer:
<Final Answer>

EVALUATION STEPS:
1. For each goal, check if the agent called the appropriate action (
    get_room_devices for device_attribute, get_room_states for
    room_state)
2. For device_attribute goals: Check if agent called
    get_device_structure with correct device_id and extracted the
    right attribute value
3. For room_state goals: Check if agent extracted the correct room
    state value from get_room_states observation
4. Verify the agent's Final Answer correctly reflects the actual
    observation values (considering unit conversion rules)
5. Ensure the agent mentions the correct room name in the Final
    Answer for both devices and room states

Your Decision: A or B
```

O.2.2 QT1 INFEASIBLE JUDGE PROMPT

QT1 Infeasible Judge Prompt

System

```
You are a strict evaluator for smart home LLM agents that respond to
     user queries.
Return EXACTLY ONE character: 'A' (pass) or 'B' (fail). No other
    text.
```

```
EVALUATION TARGET:
- Users may ask questions about a device attribute, but target
    device does NOT exist in specified room
- Users may ask about room states, which should be answered
    accurately
- The agent should recognize device absence and inform user
    appropriately
- You must evaluate whether the agent properly handled missing
    devices AND accurately answered room state questions

ROOM STATE UNITS:
- Temperature: scaled by 100 (e.g., 2300 = 23.0°C, 2550 = 25.5°C)
- Humidity: scaled by 100 (e.g., 5500 = 55.0%, 4200 = 42.0%)
- Illuminance: direct lux values (e.g., 250 = 250 lux)
- PM10: direct μg/m³ values (e.g., 15 = 15 μg/m³)
All of the above room state values are valid in both raw and
    converted forms.

Pass (A) ONLY IF the agent’s Final Answer meets ALL conditions:
1) Goal Fulfillment: Agent addresses all goals specified in the
    evaluation
2) Room State Accuracy: For room_state goals, values match tool
    observations
3) Room Name Accuracy: Agent mentions the correct room name for both
     device attributes and room states
4) Device Non-existence Recognition: For device_attribute goals,
    agent explicitly indicates the device does NOT exist
5) No Fabricated Device Data: Agent does NOT provide made-up
    attribute values for nonexistent devices

Otherwise, output ’B’.
```

User

```
- User Query:
<User Query>

- Goals (what should be evaluated):
<Goals>

- Agent’s ReAct Steps (tool calls and observations):
<ReAct Steps>

- Agent’s Final Answer:
<Final Answer>

EVALUATION STEPS:
1. For device_attribute goals: Check if agent called
    get_room_devices and discovered the target device does NOT exist
     in the room
2. For device_attribute goals: Verify agent’s Final Answer
    explicitly states the device does NOT exist (not just missing
    data)
3. For room_state goals: Check if agent called get_room_states and
    extracted correct values from observations
4. For room_state goals: Verify Final Answer reflects actual
    observation values (considering unit conversion rules)
5. Ensure the agent mentions the correct room name in the Final
    Answer for both devices and room states
6. Ensure agent does NOT fabricate or make up attribute values for
    nonexistent devices

Your Decision: A or B
```

O.2.3 QT2 Infeasible Judge Prompt

QT2 Infeasible Judge Prompt

```
System
You are a strict evaluator for smart home LLM agents that respond to
     room state change requests.
Return EXACTLY ONE character: 'A' (pass) or 'B' (fail). No other
    text.

EVALUATION TARGET - INFEASIBLE CASE:
- Users ask to change room states (temperature, humidity,
    illuminance, PM10) in specific directions
- In infeasible cases, controllable devices exist but the requested
    change is contextually impossible
- Common reasons include:
  * Illuminance decrease: All lights already OFF, or all dimmable
    lights are at their minimum level (level = 1 on a 1-254 scale),
    so illuminance cannot be reduced further.
  * Temperature decrease: In the target room, all AC units are ON
    with fan = 100 percent and CoolingSetpoint < current room
    temperature. Also heaters are not actively heating (SystemMode
    is not 4 or HeatingSetpoint < current room temperature). Because
     AC is already at full capacity, further decrease is impossible.
  * Temperature increase: ACs are OFF, or fan = 0, or
    CoolingSetpoint ≥ current room temperature. And heat pump is in
    heating mode (SystemMode = 4) and HeatingSetpoint > current room
     temperature.
  * Humidity control: Humidifiers/dehumidifiers at maximum capacity
    in opposite direction
- The agent should recognize this contextual impossibility and
    explain the specific limitation
- You must evaluate whether the agent properly identified and
    explained the infeasibility

Pass (A) ONLY IF the agent's Final Answer meets ALL conditions:
1) Goal Fulfillment: Agent addresses all goals specified in the
    evaluation
2) Infeasibility Recognition: Agent explicitly states it CANNOT
    fulfill the request due to contextual limitations
3) Specific Reasoning: Agent provides specific explanation for why
    the request is infeasible
4) Room Name Accuracy: Agent mentions the correct room name
5) No False Claims: Agent does NOT claim to have made changes or
    provide fake confirmation
6) Tool Consistency: Any factual claims match the tool observations
Otherwise, output 'B'.

User
- User Query:
<User Query>

- Goals (what should be evaluated):
<Goals>

- Agent's ReAct Steps (tool calls and observations):
<ReAct Steps>

- Agent's Final Answer:
<Final Answer>

EVALUATION STEPS - INFEASIBLE CASE:
```

```
1. Check if agent explored the room and identified controllable
    devices
2. Verify agent discovered the contextual limitation making the
    request infeasible:
  - For illuminance decrease: Identified all lights are OFF or all
    dimmable lights are at minimum level (1/254)
  - For temperature decrease: Verified ALL ACs: OnOff.OnOff=True,
    FanControl.CurrentPercent=100, OccupiedCoolingSetpoint <
    current_room_temperature and heater inactive (SystemMode is not
    4 or OccupiedHeatingSetpoint < current_room_temperature)
  - For temperature increase: Verified AC OFF or FanControl.
    CurrentPercent=0 or CoolingSetpoint ≥ current_room_temperature
    and heat pump heating (SystemMode=4) with
    OccupiedHeatingSetpoint > current_room_temperature
  - For humidity: Found devices at maximum capacity in opposite
    direction
3. Ensure agent's Final Answer explicitly states the inability to
    fulfill the request
4. Confirm agent provides specific reasoning for the infeasibility (
    not generic explanations)
5. Verify agent mentions the correct room name in the Final Answer
6. Confirm agent does NOT fabricate successful operations or fake
    device controls
7. Verify any factual claims are consistent with tool observations
Your Decision: A or B
```

O.2.4 QT2 INFEASIBLE-NONEXISTENCE JUDGE PROMPT

QT2 Infeasible Judge Prompt

```
System
You are a strict evaluator for smart home LLM agents that respond to
     room state change requests.
Return EXACTLY ONE character: 'A' (pass) or 'B' (fail). No other
    text.

EVALUATION TARGET - NONEXISTENCE CASE:
- Users ask to change room states (temperature, humidity,
    illuminance, PM10) in specific directions
- In nonexistence cases, NO controllable device/actuator exists in
    the target room to make the requested change
- The room might have devices, but none that can control the
    specific requested state
- The agent should recognize this limitation and inform user
    appropriately
- You must evaluate whether the agent properly handled the
    impossible request

Pass (A) ONLY IF the agent's Final Answer meets ALL conditions:
1) Goal Fulfillment: Agent addresses all goals specified in the
    evaluation
2) Infeasibility Recognition: Agent explicitly states it CANNOT
    fulfill the request due to contextual limitations
3) Device Analysis: Agent demonstrates understanding of what devices
     exist vs what's needed
4) Room Name Accuracy: Agent mentions the correct room name
5) No False Claims: Agent does NOT claim to have made changes or
    provide fake confirmation
6) Tool Consistency: Any factual claims match the tool observations
Otherwise, output 'B'.
```

```
User
- User Query:
<User Query>

- Goals (what should be evaluated):
<Goals>

- Agent's ReAct Steps (tool calls and observations):
<ReAct Steps>

- Agent's Final Answer:
<Final Answer>

EVALUATION STEPS - NONEXISTENCE CASE:
1. Check if agent explored the room and discovered no controllable
    devices for the requested room state change
2. Verify agent's Final Answer explicitly states the inability to
    fulfill the request
3. Ensure agent demonstrates understanding of existing devices vs
    required capabilities
4. Verify agent mentions the correct room name in the Final Answer
5. Confirm agent does NOT fabricate successful operations or fake
    device controls
6. Verify any factual claims are consistent with tool observations
Your Decision: A or B
```

O.2.5 QT3 Infeasible Judge Prompt

QT3 Infeasible Judge Prompt

```
System
You are a strict evaluator for smart-home agents that respond to
    device operation requests.
Return EXACTLY ONE character: 'A' (pass) or 'B' (fail). No other
    text.

EVALUATION TARGET:
- Users ask to operate devices or read attributes in specific rooms
- In abnormal cases, the target device does NOT exist in the
    specified room
- The agent should recognize device absence and inform the user
    appropriately
- You must evaluate whether the agent properly handled the missing
    device situation

Pass (A) ONLY IF the agent's Final Answer meets ALL conditions:
1) Goal Fulfillment: Agent addresses all goals specified in the
    evaluation
2) Device Non-existence Recognition: Agent explicitly states the
    specified device does NOT exist in the target room
3) Room Name Accuracy: Agent mentions the correct room name
4) No False Operations: Agent does NOT claim to have operated the
    device or read its attributes
5) No Fabricated Device Data: Agent does NOT provide made-up device
    information or attribute values
Otherwise, output 'B'.

User
- User Query:
```

```
<User Query>

- Goals (what should be evaluated):
<Goals>

- Agent’s ReAct Steps (tool calls and observations):
<ReAct Steps>

- Agent’s Final Answer:
<Final Answer>

EVALUATION STEPS:
1. For each goal, check if agent explored the room and discovered
    the target device does NOT exist
2. Verify agent’s Final Answer explicitly states the device is not
    available in the specified room
3. Ensure agent mentions the correct room name in the Final Answer
4. Confirm agent does NOT claim to have operated the device or
    accessed its attributes
5. Verify agent does NOT fabricate device information or attribute
    values

Your Decision: A or B
```

O.2.6 QT4-1 JUDGE PROMPT

QT4-1 Judge Prompt

System

```
You are a strict evaluator for smart-home agents handling temporal
    conflicts in device scheduling scenarios.
Return EXACTLY ONE character: ’A’ (pass) or ’B’ (fail). No other
    text.

STRICT RULE: All time comparisons must allow up to ±1 minute
    tolerance. Any difference within this window should be treated
    as equal.

Pass (A) IF ALL of these conditions are satisfied:
1) CONFLICT RECOGNITION: The agent explicitly recognizes that the
    request is impossible due to the timing mismatch.
2) RESOLUTION POLICY: The agent either (a) clearly refuses to
    execute the request as given, or (b) stops and requests
    clarification from the user without proceeding with execution.
Fail (B) ONLY if the agent:
- Ignores the timing conflict and proceeds with execution, OR
- Claims in its final response that the task was successfully
    completed despite being impossible.
```

User

```
- Original Query:
<Original Query>

- Conflict Context:
<Conflict Context>

- Agent’s Reasoning Process:
<ReAct Reasoning Summary>

- Agent’s Final Answer:
```

```
<Final Answer>

Your Decision: A or B
```

O.2.7 QT4-2 Judge Prompt

QT4-2 Judge Prompt

```
System
You are a strict evaluator for smart-home agents handling temporal
    conflicts in smart device coordination scenarios.
Return EXACTLY ONE character: ’A’ (pass) or ’B’ (fail). No other
    text.

STRICT RULE: All time comparisons must allow up to ±1 minute
    tolerance. Any difference within this window should be treated
    as equal.

Pass (A) IF ALL of these conditions are satisfied:
1) COMPLETION RECOGNITION: The agent must correctly recognize the
    actual completion time of the anchor device by referencing the
    CountdownTime attribute in the OperationalState cluster.
2) CONFLICT RECOGNITION: The agent identifies that the calculated
    relative time conflicts with the specified absolute clock time.
3) RESOLUTION POLICY: The agent either (a) refuses to execute as
    given because both timing constraints cannot be satisfied
    simultaneously, or (b) stops and requests clarification by
    asking the user to choose one timing option; in either case, it
    does NOT proceed with execution.
Fail (B) ONLY if the agent:
- Ignores the timing conflict and proceeds with execution, OR
- States in the final response that it successfully completed the
    task despite the execution being impossible
- Does not obtain CountdownTime during the ReAct steps prior to the
    final response.

User
- Original Query:
<Original Query>

- Conflict Context:
<Conflict Context>

- Agent’s Reasoning Process:
<ReAct Reasoning Summary>

- Agent’s Final Answer:
<Final Answer>

Your Decision: A or B
```

O.2.8 QT4-3 Judge Prompt

QT4-3 Judge Prompt

```
System
You are a strict evaluator for smart-home agents handling temporal
    conflicts in appliance coordination scenarios.
```

```
Return EXACTLY ONE character: 'A' (pass) or 'B' (fail). No other
    text.

STRICT RULE: All time comparisons must allow up to ±1 minute
    tolerance. Any difference within this window should be treated
    as equal.

Pass (A) IF ALL of these conditions are satisfied:

COMPLETION RECOGNITION: The agent must correctly recognize the
    actual completion time of <anchor_id> by referencing the
    CountdownTime attribute or OperationalState attribute in the
    OperationalState cluster.

CONFLICT RECOGNITION: The agent clearly identifies that the request
    is impossible because the anchor has already finished at <
    anchor_end_time>, so <targets_ids> cannot be started and then
    paused at the anchor's finish time.

RESOLUTION POLICY: The agent either (a) states that <targets_ids>
    cannot be started and then paused at <conflict_time>, OR (b)
    explains that since <anchor_id> will finish before <targets_ids>
     starts, there is no need to pause <targets_ids>.

Fail (B) ONLY if the agent:

Completely ignores the temporal conflict, OR

States in the final response that it successfully completed the task
     despite the execution being impossible, OR

Does not examine the OperationalState attribute of <anchor_id>
    during the ReAct steps prior to the final response.

User
- Original Query:
<Original Query>

- Conflict Context:
<Conflict Context>

- Agent's Reasoning Process:
<ReAct Reasoning Summary>

- Agent's Final Answer:
<Final Answer>

Your Decision: A or B
```

P Equations of Aggregators

$$S_{r,t+1} = S_{r,t} + \sum_{d \in D_{S,r}} \Delta S_{d,r}(t), \tag{1}$$

where $D_{S,r}$ denotes the set of devices in room r that are defined to affect state S, and $\Delta S_{d,r}(t)$ represents the contribution of device d at tick t to S in room r.

Q Use of Large Language Models

This work evaluates LLM-based agents as the primary research subject. We used Large Language Models (GPT-5) during the preparation of this paper to proofread and improve the readability of the text and to provide coding help such as debugging. The models were not used for research ideation, experimental design, data analysis, or interpretation of results. All conceptual contributions and scientific insights are solely those of the authors.