HomeBench: Evaluating LLMs in Smart Homes with Valid and Invalid Instructions Across Single and Multiple Devices

Silin Li[1], Yuhang Guo[1], Jiashu Yao[1], Zeming Liu[2] *, Haifeng Wang[3]
[1]School of Computer Science and Technology, Beijing Institute of Technology
[2]School of Computer Science and Engineering, Beihang University [3]Baidu Inc.
{lisilin,guoyuhang,yaojiashu}@bit.edu.cn zmliu@buaa.edu.cn wanghaifeng@baidu.com

Abstract

Large language models (LLMs) have the potential to revolutionize smart home assistants by enhancing their ability to accurately understand user needs and respond appropriately, which is extremely beneficial for building a smarter home environment. While recent studies have explored integrating LLMs into smart home systems, they primarily focus on handling straightforward, valid single-device operation instructions. However, real-world scenarios are far more complex and often involve users issuing invalid instructions or controlling multiple devices simultaneously. These have two main challenges: LLMs must accurately identify and rectify errors in user instructions and execute multiple user instructions perfectly. To address these challenges and advance the development of LLM-based smart home assistants, we introduce HomeBench, the first smart home dataset with valid and invalid instructions across single and multiple devices in this paper. We have experimental results on 13 distinct LLMs; e.g., GPT-4o achieves only a 0.0% success rate in the scenario of invalid multi-device instructions, revealing that the existing state-of-the-art LLMs still cannot perform well in this situation even with the help of in-context learning, retrieval-augmented generation, and fine-tuning. Our code and dataset are publicly available at the link[1].

1 Introduction

Integrating large language models (LLMs) into smart home assistants can not only help users control devices with simpler and fewer instructions but also enable automatic device control based on the user's habits and history actions, without the need for explicit instructions (King et al., 2024; Rivkin et al., 2024; Yin et al., 2024). For instance, King et al. (2024); Rivkin et al. (2024) introduced

*Corresponding author: Zeming Liu
[1]https://github.com/BITHLP/HomeBench

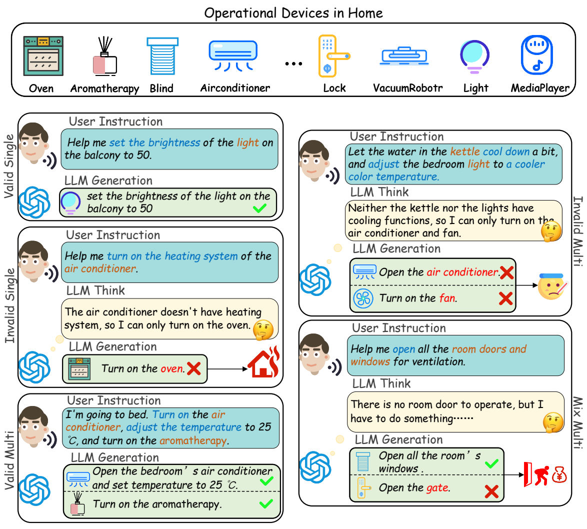
Operational Devices in Home
Oven
Aromatherapy
Blind
Airconditioner
...
Lock
VacuumRobotr
Light
MediaPlayer
Valid Single
User Instruction
Help me set the brightness of the light on the balcony to 50.
LLM Generation
set the brightness of the light on the balcony to 50
Invalid Single
User Instruction
Help me turn on the heating system of the air conditioner.
LLM Think
The air conditioner doesn't have heating system, so I can only turn on the oven.
LLM Generation
Turn on the oven.
Valid Multi
User Instruction
I'm going to bed. Turn on the air conditioner, adjust the temperature to 25 ℃, and turn on the aromatherapy.
LLM Generation
Open the bedroom' s air conditioner and set temperature to 25 ℃.
Turn on the aromatherapy.
Invalid Multi
User Instruction
Let the water in the kettle cool down a bit, and adjust the bedroom light to a cooler color temperature.
LLM Think
Neither the kettle nor the lights have cooling functions, so I can only turn on the air conditioner and fan.
LLM Generation
Open the air conditioner.
Turn on the fan.
Mix Multi
User Instruction
Help me open all the room doors and windows for ventilation.
LLM Think
There is no room door to operate, but I have to do something……
LLM Generation
Open all the room' s windows .
Open the gate.

Figure 1: An example of valid single-device instruction, invalid single-device instruction, valid multi-device instruction, invalid multi-device instruction, and mixed multi-device instruction. Orange indicates the device to be operated, blue indicates the method of the device to be operated, and red indicates the wrong device to be operated. These mistake outputs are due to the operation of non-existent devices or non-existent device functions, causing the model to hallucinate and operate the wrong device.

the Sasha and Sega systems, which preliminarily validated the feasibility of integrating LLMs into smart home assistants. Yin et al. (2024) further explored the capability of LLMs in handling ambiguous and no-instruction scenarios for smart home control, providing additional evidence for the promising application of large language models in this field. These researches primarily focus on the operation of valid single-device instructions (King et al., 2024; Rivkin et al., 2024; Yin et al., 2024), and the range and number of devices that LLMs can control are quite limited in these researches.

However, in real-world scenarios, users may input invalid instructions for various reasons, such as being intoxicated or misspeaking. If the model faithfully executes these invalid instructions, it may negatively affect the user experience and safety. As Figure 1 shows, mistakenly activating kitchen appliances or unlocking doors could pose significant safety risks. In addition, in real-world scenarios that involve multiple devices, users often need to control several devices simultaneously. Therefore, the development of LLMs capable of comprehensively managing multiple devices and understanding dynamic interactions in complex environments, while optimizing mechanisms for warning and correcting potential errors, becomes a key factor in the advancement of smart home systems.

To bridge these gaps, we introduced a new comprehensive evaluation benchmark: HomeBench, which is the first smart home dataset with valid and invalid instructions across single and multiple devices, with a diverse range and a large number of devices. Additionally, to facilitate flexible experimentation and scenario expansion, we developed a customizable virtual home environment that allows for the flexible setup of various devices and operational scenarios, offering a versatile experimental base for model testing and optimization. Specifically, we constructed 100 smart home scenarios covering more than 170k user operation instructions. Each scenario includes at least 47 smart devices across a minimum of 10 different types. For multi-device operation scenarios, we simulated complex interactions involving 1 to 10 devices to gauge the model's performance in highly intricate environments. By introducing HomeBench, we aim to pave the way for next-generation smart home assistants capable of seamlessly managing and coordinating numerous smart devices within modern IoT ecosystems.

Our main contributions can be summarized in the following three points:

- To the best of our knowledge, we are the first to study the model's ability in invalid instructions and multi-device operation instructions in smart homes.
- We proposed HomeBench, which contains valid and invalid instructions across single and multiple devices, covers complex operation scenarios, and developed a customizable virtual environment to support the generation and expansion of datasets.
- Our experimental results on 13 different LLMs show that almost all models perform poorly in this benchmark, especially on invalid instructions and multi-device instructions. Further analysis shows that simple in-context learning and fine-tuning significantly improve performance, but there is still a gap between them and practical application scenarios.

2 Related Work

2.1 Home Assistant before LLM

Smart home systems are built from interconnected IoT smart devices that monitor, sense, and control the home environment. Automating these devices not only improves quality of life and comfort but also optimizes resource efficiency (Ki et al., 2020). Recent researches focus on enhancing the functionality of smart home systems through machine learning techniques. For example, Jaihar et al. (2020) develop a deep learning algorithm to identify user behavior using accelerometer data to achieve home automation. Rani et al. (2017) focus on voice-based home assistants, which are designed to understand user voice instructions. Using advanced NLP techniques, commercial solutions such as Bixby, Google Assistant, and Alexa provide user-friendly interfaces that can handle a variety of instructions and inquiries from shopping and setting reminders to device control and home automation. However, these modern home assistants still face challenges in handling ambiguous or complex instructions (Luger and Sellen, 2016).

2.2 Home Assistant on LLM

In an attempt to overcome some of these challenges, recent work proposes using the reasoning capabilities of LLMs to better understand and carry out user instructions. Shi et al. (2024a) propose AwareAuto, a system that standardizes user expressions and employs a two-step reasoning method powered by LLMs to generate automation solutions, achieving a 91.7% accuracy rate in aligning with user intents. Civitarese et al. (2024) develope ADL-LLM, which transforms raw sensor data into textual descriptions to enable zero-shot recognition of daily living activities. GestureGPT (Zeng et al., 2024) integrates LLM reasoning capabilities into a triple-agent framework, enabling gesture analysis and execution of smart home tasks. Sasha (King et al., 2024) showcases the application of LLMs in smart home environments, demonstrating their ability to respond effectively to complex or ambiguous

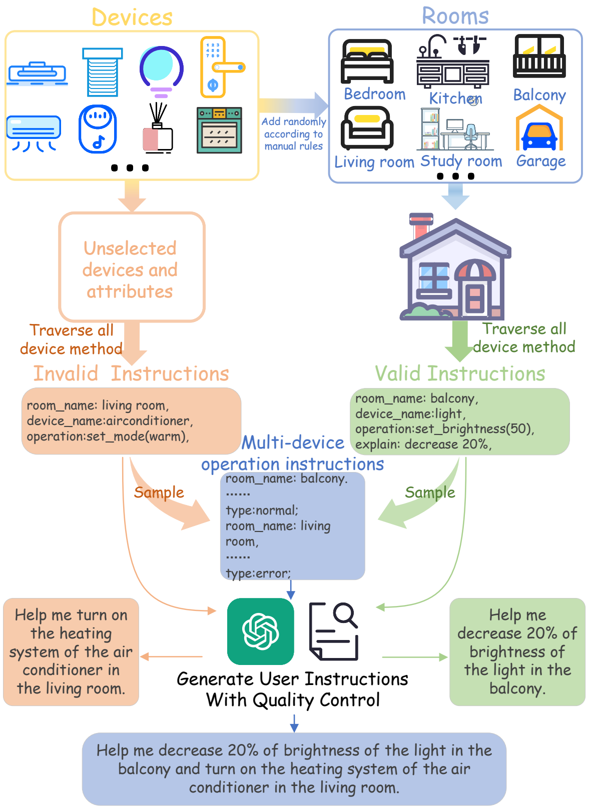
Devices
Rooms
Bedroom
Kitchen
Balcony
Living room
Study room
Garage
Add randomly according to manual rules
Unselected devices and attributes
Traverse all device method
Traverse all device method
Invalid Instructions
Valid Instructions
room_name: living room,
device_name:airconditioner,
operation:set_mode(warm),
room_name: balcony,
device_name:light,
operation:set_brightness(50),
explain: decrease 20%,
Multi-device operation instructions
room_name: balcony.
......
type:normal;
room_name: living room,
......
type:error;
Sample
Sample
Help me turn on the heating system of the air conditioner in the living room.
Help me decrease 20% of brightness of the light in the balcony.
Generate User Instructions With Quality Control
Help me decrease 20% of brightness of the light in the balcony and turn on the heating system of the air conditioner in the living room.

Figure 2: The whole process of collecting dataset HomeBench, from device construction, room setting, instructions generation, and user instructions synthesis to quality control.

instructions. Sasha implements a decision-making process where each step (e.g., device selection or routine verification) is supported by LLMs. The SAGE system addresses the limitations of LLMs in lacking specific knowledge about users and households by dynamically constructing prompt trees and employing LLM-controlled discrete operation sequences. This significantly enhances its capabilities in information retrieval, user interaction, and device operation. However, these researches ignore the real scenarios of user invalid instructions and multiple operation instructions.

3 Dataset Collection

HomeBench aims to build high-quality smart home instruction call data to improve the single-device and multi-device call capabilities in complex smart home environments, focusing on the accurate response capabilities of LLM when facing user invalid instructions. In this section, we describe the dataset task format and then explain how to build the dataset efficiently and effectively. Figure 2 shows the whole process of collecting dataset HomeBench.

3.1 Task Definition

Given a user instruction u and a virtual home environment $H = \{r_1, r_2, r_3, \dots\}$ with room information, each room r_i contains a set of operable devices $D = \{d_1, d_2, \dots\}$ in the current room. Each device d_j includes the state of its operable properties s and a set of callable methods $M = \{m_1, m_2, m_3, \dots\}$ and corresponding parameters. The LLM needs to decide which method of which device in which room to call to change the properties of the target device based on the user instruction u, that is, $h, u \rightarrow [r_i.d_j.m_k]$. If the user inputs an invalid instruction, the output is error_input.

3.2 Data Categorization

Dataset	VS	IS	VM	IM	MM	SIZE
IFTTT (Yu et al., 2021)	✓	✗	✓	✗	✗	50K+
SEGA (Rivkin et al., 2024)	✓	✗	✗	✗	✗	50
Home Assistant[2]	✓	✗	✗	✗	✗	30K+
Ours	✓	✓	✓	✓	✓	170K+

Table 1: Comparison with existing dataset at the instruction category for a fair comparison. "✓" represents the automated instruction generation for smart home operations, which differs significantly from our task setting.

Based on the types of instructions and the number of devices involved, we can categorize the data into five distinct types. Each type aligns with a typical use case in real-world scenarios, providing a comprehensive benchmark for assessing the capabilities of LLM in invoking smart home devices:

- Valid single-device instruction (VS): The user issues an instruction that involves a single method of a single device. The instruction is correct and executable.
- Invalid single-device instruction (IS): The user issues an instruction that involves a single method of a single device. However, the instruction is incorrect and cannot be executed.
- Valid multi-device instruction (VM): The user issues instructions involving multiple methods across multiple devices. All instructions are correct and executable.

[2]https://huggingface.co/datasets/acon96/Home-Assistant-Requests

Statistics		VS	IS	VM	IM	MM	ALL
Train	Avg. Instruction Length	11.13	11.27	32.48	27.82	69.37	25.59
	Avg. Number of Operating Device	1.00	1.00	2.91	2.51	6.26	2.30
	Number of Instructions	49,048	53,771	1,829	826	33,443	138,917
Valid	Avg. Instruction Length	11.10	11.29	34.37	26.88	69.30	25.45
	Avg. Number of Operating Device	1.00	1.00	3.12	2.40	6.26	2.26
	Number of Instructions	6,145	6,731	232	124	4,132	17,364
Test	Avg. Instruction Length	11.19	11.28	33.49	25.88	68.81	25.13
	Avg. Number of Operating Device	1.00	1.00	2.98	2.30	6.21	2.29
	Number of Instructions	6,187	6,765	245	97	4,072	17,366

Table 2: The data statistics of our proposed `HomeBench`.

- Invalid multi-device instruction (IM): The user issues instructions involving multiple methods across multiple devices. However, all instructions are incorrect and cannot be executed.

- Mix multi-device instruction (MM): The user issues instructions involving multiple methods across multiple devices. Some instructions are correct and executable, while others are incorrect and cannot be executed.

Table 1 shows a detailed comparison of the dataset `HomeBench` with other popular datasets. Currently, most benchmarks focus on processing valid instructions of a single device, while common multi-device operation instructions and invalid instructions in real life are often ignored. This study makes important contributions in these two aspects.

3.3 Data Collection

Given the challenges in accessing smart home development environments on the market, we adopted a virtual smart home setting to reduce the entry barrier for our research. The dataset construction is divided into two primary phases: building virtual devices and environments and generating instructions.

For virtual device construction, we collected 15 commonly used smart home devices and identified the API methods that can be called through the smart home platform for each device. For example, devices such as robotic vacuum cleaners. We implemented these virtual devices using Python classes, allowing property modifications through API method calls. To account for performance differences among various brands and models, we simplified the methods for some devices, retaining only the essential smart home functions. For other devices, we randomly added additional methods to create a diverse pool of devices. To increase the complexity of the task, we designed twelve rooms, each with unique functionalities. Devices were randomly selected from the pool and assigned to rooms based on manually defined rules.

To ensure data quality while minimizing manual effort, we used GPT-3.5 to generate user instructions (Wang et al., 2024d). For valid instructions, we first created all possible device operation commands within the virtual environment, specifying the room, device name, method called, and the method's input parameters. With the help of prompt rules and a few-shot examples, this information was provided to ChatGPT to synthesize user instructions. For invalid instructions, we applied reverse rules to identify non-existent devices or missing methods within the virtual room and generated corresponding erroneous operation commands. These instructions were also processed through ChatGPT using the same approach. For scenarios involving multiple instructions, we randomly selected commands from both the pool of valid operations and the invalid instruction pool. We then used the same synthesis method to generate user instructions.

To ensure the quality of the data, we sampled 10% of the user instructions generated in each smart home scenario and conducted a rigorous evaluation from three dimensions: fluency checks (Wang et al., 2024b; Liu et al., 2020) (language clarity), semantic alignment (Iskander et al., 2024) (instruction vs. device/room capabilities) and action consistency (Iskander et al., 2024) (synthesized instructions vs. executable machine instructions). Any mismatch resulted in immediate rejection (score=0). If an instruction scored more than 4, the data of the scenario was considered valid. This process was repeated until 100 high-quality smart home scenario data were collected.

3.4 Data Statistic and Analysis

Table 2 shows the statistics of the dataset. Following previous works (Zhang et al., 2023; Kweon et al., 2023; Cheng et al., 2022), we split the dataset in the conventional 8:1:1 ratio and provide detailed information on the average instruction length, number of instruction operation devices, and number of instructions according to the data types of VS, IS, VM, IM, and MM. Our data distribution reveals two notable patterns that warrant explanation: (1) the prevalence of single-device instructions over multi-device instructions, and (2) the relatively small number of IM instructions. Below we provide detailed explanations for these observations:

- **Single-device instructions nums > multi-device instructions nums:** From the perspective of load balancing theory, humans tend to execute a single type of task multiple times rather than perform multiple distinct tasks simultaneously (Sweller, 2011). As a result, single-device operation commands are significantly more frequent than multi-device operation commands.
- **Small number of IM instructions:** From a probabilistic standpoint, the likelihood of executing entirely valid or entirely invalid operations across multiple devices is much lower than that of mixed valid and invalid operations.

To demonstrate the complexity of our task, we statistic the types and numbers of devices in 100 homes. In 12 rooms with different functions, at least 47 devices can be operated, and at least 10 types of devices can be operated. More analysis can be found in the Appendix A.1.

4 Experiment

4.1 Setting

Models. Inspired by Fei et al.; Huang et al., we choose several LLMs from both open- and closed-source models, aiming to provide a comprehensive evaluation. Specifically, we choose Mistral-7B (Mistral-7B-v0.3) (Jiang et al., 2023), LLaMa3-8b (llama3-8b-Instruct) (AI@Meta, 2024), the Qwen series (Qwen2.5-7B/32B/72B-Instruct) (Yang et al., 2024), the Gemma series (gemma-2-9b/27b-it) (Team et al., 2024) from opensource LLMs. Besides that, we also select GPT4 (GPT-4o) and deepseek v3 (DeepSeek-AI et al., 2024) from closed-source LLMs. In addition, we also test reasoning-oriented models such as O1,O3 (o1-mini, o3-mini) (OpenAI et al., 2025), DeepSeek-R1 (DeepSeek-AI et al., 2025), and QWQ-32B(Team, 2025). We built two types of baseline models for experimental comparison: one is based on prompt, and the other is based on few-shot in-context learning (Brown et al., 2020; Zhou et al., 2024). As shown in Sec 5.1, the performance improvement of the model after 3-shot is limited, so we chose 4-shot in the experiment. In both types of baseline models, we embed the status information of each device in the virtual home and the methods that the device can call into the prompts to provide contextual information for the model. This setting is designed to ensure that the two methods are evaluated under the same information conditions to comprehensively compare their performance. For specific details, please refer to Appendix A.2.

4.2 Evaluation Metrics

In the experimental design of this study, we adhered to standardized testing paradigms in the field of embodied intelligence, employing a virtual environment as the test platform (Zhu et al., 2024; Zhang et al., 2024b). Based on the principle of functional equivalence, the virtual and physical devices maintain fully consistent testing criteria at the API interface level: a test is deemed passed when the model correctly generates API call sequences that comply with predefined specifications. Notably, the virtual testing environment offers unique experimental advantages, as it eliminates mechanical delays inherent to physical devices (by removing the need to wait for real-world feedback cycles), thereby significantly enhancing testing efficiency.

To evaluate the effectiveness of LLM in executing user instructions, we selected two key indicators: instruction execution success rate (Shi et al., 2024a; Rivkin et al., 2024) and F1 score.

Success Rate (Succ): We follow Shi et al.; Rivkin et al., this metric is an indicator to measure whether all device operations in a user instruction are executed successfully.

F1 (Li et al., 2024; Cai et al., 2024; Wang et al., 2024c): We first calculate the precision P, which is defined as the number of operations correctly predicted by the model divided by the total number

model	VS		IS		VM		IM		MM		ALL	
	SUCC	F1	SUCC	F1	SUCC	F1	SUCC	F1	SUCC	F1	SUCC	F1
Mistral-7B	2.49	4.68	0.01	0.03	0.82	6.23	0.00	0.00	0.00	2.26	0.90	2.33
LLaMA3-8B	47.11	47.36	0.61	0.61	3.27	31.83	0.00	0.52	0.00	15.20	17.07	18.21
Qwen2.5-7B	30.40	31.92	0.00	0.00	13.69	33.81	0.00	0.00	0.00	16.13	11.02	16.24
Qwen2.5-32B	41.28	41.34	14.04	16.34	14.29	26.33	0.00	9.55	0.27	19.45	20.44	23.06
Qwen2.5-72B	42.90	42.91	11.47	12.57	19.59	38.06	1.03	9.07	0.15	24.79	20.07	25.92
Gemma2-9B	12.32	12.58	0.00	0.02	0.41	11.44	0.00	0.98	0.00	6.40	4.39	6.42
Gemma2-27B	12.32	13.08	0.93	1.03	0.82	11.18	0.00	2.33	0.02	6.93	4.77	7.34
Deepseek-V3	71.08	71.08	39.05	38.40	45.31	69.14	0.00	11.11	0.50	28.15	41.19	38.81
GPT-4o	46.74	46.76	79.07	78.90	13.88	26.74	0.00	55.19	2.09	28.43	48.14	42.70
QWQ-32B	63.39	63.60	7.35	11.05	6.95	15.88	12.35	30.00	3.54	32.20	26.53	35.30
Deepseek-R1	65.67	65.76	83.74	83.76	0.00	1.95	0.00	23.22	1.41	23.22	56.39	40.96
o1-mini	38.65	39.46	20.34	30.34	1.19	2.09	0.00	27.75	0.73	15.13	21.85	23.20
o3-mini	27.57	28.16	45.43	57.51	4.56	13.64	0.00	33.08	0.21	16.30	27.66	27.32
Mistral-7B-ICL	47.10	53.63	0.15	0.17	17.96	50.07	0.00	0.00	0.02	22.74	17.10	24.15
LLaMA3-8B-ICL	68.08	68.17	2.38	2.37	30.61	60.63	0.00	2.71	0.17	28.69	25.65	30.78
Qwen2.5-7B-ICL	66.96	66.89	14.07	14.08	37.55	69.18	0.00	5.82	0.49	33.23	29.98	35.72
Qwen2.5-32B-ICL	77.24	77.48	69.33	69.27	45.31	73.46	3.09	44.44	5.38	52.71	56.44	59.84
Qwen2.5-72B-ICL	82.27	82.07	35.77	35.78	51.02	76.05	6.19	27.89	2.95	48.42	44.60	52.01
Gemma2-9B-ICL	59.72	59.65	21.67	21.74	33.47	62.54	3.09	20.62	1.55	42.17	30.57	41.67
Gemma2-27B-ICL	79.44	79.50	11.06	11.07	48.16	74.71	1.03	11.21	0.81	41.97	33.48	43.00
Deepseek-V3-ICL	80.00	80.06	58.58	58.48	54.69	77.29	11.34	32.22	5.35	52.74	53.41	58.74
GPT-4o-ICL	74.25	74.30	87.10	87.09	45.71	71.48	61.86	81.18	23.35	78.98	66.85	79.58
QWQ-32B-ICL	67.71	64.50	79.32	79.30	35.51	61.97	65.98	79.39	18.71	70.02	60.28	70.50
Deepseek-R1-ICL	76.48	77.46	84.18	84.77	46.19	70.75	66.27	83.03	28.28	77.73	67.62	78.82
o1-mini-ICL	75.42	75.94	86.80	86.90	53.39	75.52	**79.17**	**89.09**	32.25	81.37	69.47	81.42
o3-mini-ICL	**83.77**	**84.25**	**88.36**	**88.38**	**57.51**	**80.49**	64.04	85.19	**38.49**	**85.57**	**74.44**	**85.75**

Table 3: The main results of different LLMs on `HomeBench`. **Bold** indicates the best performance.

of operations predicted by the model.

$$Precision = \frac{\text{operation\_correct\_num}}{\text{operation\_pred\_num}} \tag{1}$$

We calculate the recall R, which is the number of operations correctly predicted by the model divided by the total number of operations actually required in the user instruction.

$$Recall = \frac{\text{operation\_correct\_num}}{\text{operation\_gold\_num}} \tag{2}$$

The F1 score is 2*PR / (P+R), as usual.

4.3 Results

Table 3 shows the results of different LLMs for different types of user instructions on `HomeBench`, respectively. Several conclusions can be drawn from the results.

Overall, o3-mini achieves the best overall performance, while o1-mini outperforms o3-mini in certain situations, especially when processing only invalid instructions. Generally, with the exception of Deepseek-V3, gpt4o and reasoning models, other models significantly lag behind o3-mini in handling various types of instructions. This phenomenon can be attributed to the core difficulty of our task, which involves long-context attention. The model is required to thoroughly interpret a complete set of device states and methods to identify operable devices and those that are nonexistent. Among the models evaluated, o3-mini exhibits a markedly superior ability to manage long-range context. Despite significant progress in LLMs, existing models still struggle with complex planning tasks, particularly when they involve hybrid instructions that require multi-device operations. In fact, even with the inclusion of ICL in MM tasks, the o3-mini success rate is only 38.49%.

As the size of the models increases, we observe improvements in performance in different types of instructions. This enhancement is particularly evident in the Qwen2.5 and Gemma2 series of models. Generally, larger models tend to deliver superior performance. However, the performance of most models significantly decreases in the presence of input invalid instructions or multi-device instructions, with some models getting 0% at SUCC on specific tasks, such as IM tasks, indicating ongoing challenges in fully recognizing errors. Even advanced models like Deepseek-v3 can only partially detect errors within the instructions. Furthermore, these issues persist even when ICL is incorporated into the task setup, with most models struggling to overcome the challenges presented by IM tasks.

With the introduction of ICL, the model demonstrated significant performance improvements in handling various instruction tasks. This phenomenon has been validated in all tested models, and some models have even achieved breakthrough performance gains. For example, the Mistral model demonstrated a more than 10-fold improvement in IS tasks, while the Gemma2-9b model saw its IS task success rate jump from 0% to 21.67%. Even for more complex tasks such as IM and MM, performance improvements were substantial. Notably, the Gemma2-9b model achieved a 20-point increase in these tasks. However, despite these significant enhancements, a considerable gap remains between the model's performance and the requirements of real-world applications, highlighting the need for further optimization and refinement.

The performance of the models is heavily influenced by the complexity of the instructions. Analyzing performance in different scenarios reveals the following trend: VS>VM>IS>MM>IM. This trend is consistent across a majority of large language models, including Qwen2.5, Gemma2, LLaMA3-8B, Deepseek-v3, and Mistral-7B. It aligns with our expectations, as IM is the most complex task, followed by MM and IS, while VM is relatively simpler, and VS is the easiest. However, this trend does not hold for Deepseek-V3, GPT-4o and reasoning models. The key reason lies in the nature of IS and IM tasks, which require strong long-context attention. In these tasks, the model must track the states and methods of all devices to determine which operations are valid and which devices are unavailable. Deepseek-V3, GPT-4o, and reasoning models outperform other open-source models in terms of context handling capabilities (Grattafiori et al., 2024; Wang et al., 2024a; Zhang et al., 2024c).

5 Analysis

In this section, we conduct a comprehensive analysis, aiming to answer three research questions. RQ1: How does the few-shot influence the model performance? (Sec 5.1) RQ2: Is it necessary to identify the room and device first to reduce the context window? (Sec 5.2) and RQ3: What is the major bottleneck of current LLMs in our task (Sec 5.3), and can fine-tuning alleviate it? (Sec 5.4).

5.1 The Effects of Different ICL

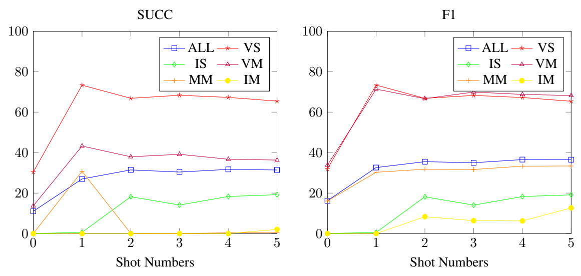

Figure 3: The performance of Qwen2.57BInstruct model in adding different types of data samples (the order of adding is: VS, IS, VM, MM, IM).

Figure 3 shows the performance in the context of Qwen2.5-7B-Instruct when using different shots at the demonstrations, the performance of other models is in the AppendixA.2. Since our data types are relatively rich, the types of shots added each time are also different. The specific order of addition is: VS, IS, VM, MM, IM. The shots here are obtained by random sampling in the training set. For detailed sampling methods and information, please refer to the description in the AppendixA.2. The experimental results show that when we add different types of shots, the overall effect of the model is improved to a certain extent, and the corresponding data types also show different degrees of improvement. For example, when adding VS (1-shot), the success rate of the model on VS is increased from 30.40% to 73.35%; and when adding IM (5-shot), the F1 score of the model at IM increases from 6.27% to 12.67%. These results show that appropriately increasing the number and type of shots can help improve the performance of the model on specific datasets. However, we also observed that as the number of shots increases, performance declines in some tasks, particularly IS and VS. This may be because the provided shots primarily focus on user instructions and device operations while lacking information about the current status of home devices. As a result, the model struggles to accurately process these tasks, leading to performance degradation. Therefore, while increasing the number of shots generally improves overall performance, it is crucial to ensure the completeness and relevance of the shot content. Failing to include key contextual information may introduce information bias, negatively impacting the model's effectiveness in specific tasks.

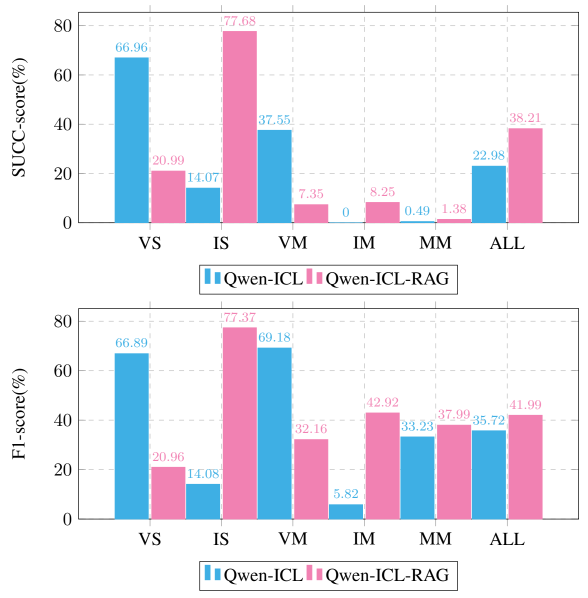

Figure 4: The performance gap between Qwen-ICL and Qwen-ICL-RAG.

5.2 The Effects of RAG

In our task setup, it is necessary to provide the status information of smart devices and the callable methods in the prompt, which results in a high token count, exceeding 3k tokens. However, each operation typically involves only 1 to 10 devices, making much of the information redundant. To reduce redundancy and improve overall performance, we attempted to optimize using the retrieval-augmented generation (RAG) (Lewis et al., 2020; Wang et al., 2024e; Shi et al., 2024b). We segmented the context based on room information, retrieving the most relevant rooms for the task at hand and providing them to the LLM. Specific details of the setup can be found in the Appendix A.2. The results are shown in Figure 4. It can be observed that the experimental results were anomalous: tasks that the model was originally proficient in showed a decline in performance, while tasks it was less proficient exhibited significant improvement. We conducted an error analysis of the results and found that the retrieved rooms contained considerable errors. Although the context was shortened, the amount of useful information was also reduced, leading to a decline in the performance of tasks involving valid instructions, while the performance of tasks involving invalid instructions improved. We have added a statistical analysis of retrieval context errors in Table 4 and output cases when retrieved incorrect context in the Appendix A.

	Single Device Instruction	Multi Devices Instructions
None Useful State Context	74.49%	46.47%
None Useful Method Context	75.97%	48.74%
Lack Useful State Context	-	46.65%
Lack Useful Method Context	-	45.04%

Table 4: Statistics on Errors in RAG Context Retrieval. We conducted a statistical analysis to determine whether the context retrieved using embedding vector similarity for the Qwen model aligns with the required context for the given input instructions.

5.3 Error Analysis

Error Type	Input	Generated and Golden	Ratio
Unfaithfulness	... master_bedroom: light: state: off ... <User instruction:> Please turn the light in the master bedroom to 50%.	error_input master_bedroom.light. set_brightness(50)	27.93

In-context Attention Error	...
	living_room:
	media_player:
	state: on
	volume:28
	...
	<User instruction:>
	Decrease the volume of the media player on the balcony by 3 percent.
		living_room.media_
		player.set_volume(20)
		living_room.media_
		player.set_volume(25)
			43.39

	... store_room: (There is no airpurifiers) ... <User instruction:> Help me jump the air purifier in the storage room to strong mode.	store_room.airpur ifiers.set_mode(high) error_input	

Key Error	...
	guest_room:
	heating:
	state:on
	mode: heating
	...
	<User instruction:>
	Set the heating in the guest bedroom to fan only mode.
		guest_bedroom.heating.
		set_mode("fan_only")
		guest_bedroom.heating
		.set_mode(fan_only)
			34.39

Table 5: The three main error types of GPT-4o test results are unfaithfulness, in-context attention error and key error, respectively. Red represents the generated green represents the golden answer. Since multi-device operations generate multiple instructions, a user instruction may contain multiple error types. We also provide the proportion of various error types for various data types in the Appendix A.2.

We performed an error analysis on the experimental results to identify potential bottlenecks in the current best model, GPT-4o. As shown in Table 5, the errors can be categorized into the following three types:

1) Unfaithfulness: The model fails to execute user instructions accurately, incorrectly judging a

correct instruction as incorrect. For example, in the case shown in the table, the model mistakenly assumes that a light is inoperable simply because the master bedroom light is turned off, leading to an incorrect response.

2) In-Context Attention Error: The model failed to notice key information contained in the context, leading to the output of invalid instructions. For instance, the user instruction in the table was to adjust the relative volume, but the model did not accurately grasp the current device's status information. Another issue is that the model did not consider the status and methods of all devices comprehensively, leading to an inability to confirm which devices were operable or non-existent.

3) Key Error: The model's output does not adhere to the prompt's specified format, preventing effective statistical analysis of the results.

5.4 The Effects of Fine-tuning

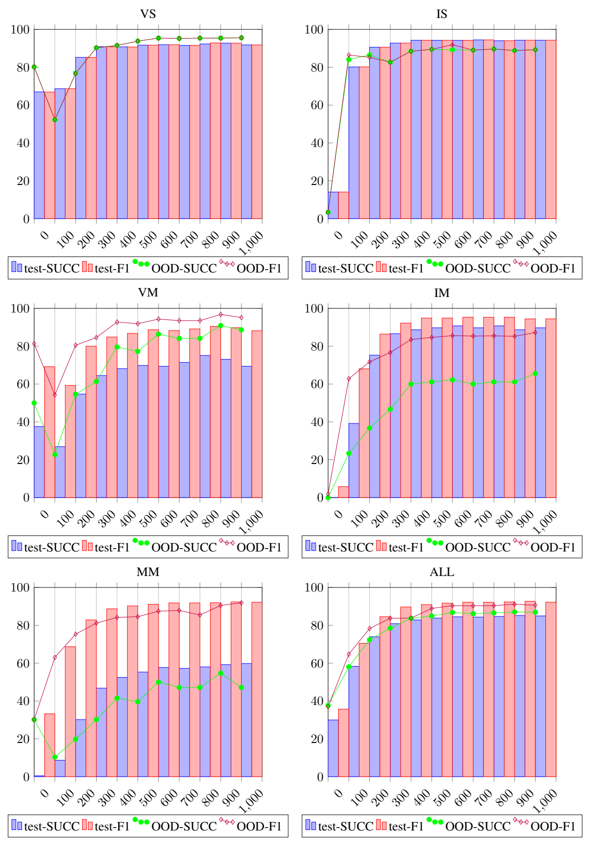

Figure 5: Performance of Qwen2.5-7B-Instruct in test dataset and OOD dataset under different training steps.

We fine-tuned Qwen2.5-7B-Instruct on the full training dataset using LoRA for 2 epochs, testing every 100 training steps. The test results are shown in Figure 5. As the training steps increased, the performance across all tasks improved significantly, but after 1 epoch of training, the performance improvement gradually plateaued. Ultimately, the model outperformed GPT-4o-ICL in all tasks. For most tasks, except VM and MM, both SUCC and F1 scores exceeded 80%. However, despite training, the SUCC scores for VM and MM remained below 70%. To further assess generalization, we conducted an Out-of-Distribution (OOD) evaluation, introducing two previously unseen devices. The results show that even on these new devices, the model's performance remained consistent with its test set results, demonstrating the robustness and diversity of our dataset. For detailed information on the OOD data, please refer to Appendix A.2. However, while fine-tuning led to performance improvements, it did not fundamentally resolve key issues. A deeper analysis revealed that the primary reasons for low SUCC scores were In-Context Attention Errors and Unfaithfulness, highlighting areas that still require further.

6 Conclusion

To advance the development of LLM-based smart home assistants and address the challenges posed by invalid instructions and multi-device instructions in real-world scenarios, we propose `HomeBench`, the first smart home dataset with valid and invalid instructions across single and multiple devices in this paper. We evaluated 13 LLMs and explored various methods to enhance their performance, including RAG, ICL, and fine-tuning. Our work lays a solid foundation for the optimization and application of LLM-based smart home assistants.

Limitations

We acknowledge the following limitations of our work. Our dataset consists of only English monolingual data, whereas real-world applications may involve multiple languages. We did not account for the differences between devices of various brands during the construction process, although such differences do exist in real-world applications. We believe that these applications can typically be distinguished by modifying device names. Moreover, the choice of brand often depends on personal preference, which is beyond the scope of this study.

Ethical Statement

During our research, we have thoroughly reviewed and ensured compliance with ethical standards. The equipment and rooms we constructed were carefully selected, and strict quality control was applied during the data synthesis process to avoid any form of offensive or biased content. Therefore, we believe there are no ethical issues with our research. The data used is ethically sourced, the analysis is fair, and all procedures adhere to established ethical guidelines.

Acknowledgement

We thank all the anonymous reviewers for their insightful and valuable comments. This work is supported by the National Natural Science Foundation of China (Grant No. U21B2009, 62406015) and Beijing Institute of Technology Science and Technology Innovation Plan (Grant No. 23CX13027).

References

AI@Meta. 2024. Llama 3 model card.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

Yuxiang Cai, Qiao Liu, Yanglei Gan, Run Lin, Changlin Li, Xueyi Liu, Da Luo, and JiayeYang JiayeYang. 2024. DiFiNet: Boundary-aware semantic differentiation and filtration network for nested named entity recognition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6455–6471, Bangkok, Thailand. Association for Computational Linguistics.

Liying Cheng, Lidong Bing, Ruidan He, Qian Yu, Yan Zhang, and Luo Si. 2022. IAM: A comprehensive and large-scale dataset for integrated argument mining tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2277–2287, Dublin, Ireland. Association for Computational Linguistics.

Gabriele Civitarese, Michele Fiori, Priyankar Choudhary, and Claudio Bettini. 2024. Large language models are zero-shot recognizers for activities of daily living. *Preprint*, arXiv:2407.01238.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, , and et al. 2024. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Zhiwei Fei, Xiaoyu Shen, Dawei Zhu, Fengzhe Zhou, Zhuo Han, Alan Huang, Songyang Zhang, Kai Chen, Zhixin Yin, Zongwen Shen, Jidong Ge, and Vincent Ng. 2024. LawBench: Benchmarking legal knowledge of large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7933–7962, Miami, Florida, USA. Association for Computational Linguistics.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, , and et al. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, and Kang Liu. 2024. DA-code: Agent data science code generation benchmark for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 13487–13521, Miami, Florida, USA. Association for Computational Linguistics.

Shadi Iskander, Sofia Tolmach, Ori Shapira, Nachshon Cohen, and Zohar Karnin. 2024. Quality matters: Evaluating synthetic data for tool-using LLMs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 4958–4976, Miami, Florida, USA. Association for Computational Linguistics.

John Jaihar, Neehal Lingayat, Patel Sapan Vijaybhai, Gautam Venkatesh, and K. P. Upla. 2020. Smart home automation using machine learning algorithms. In *2020 International Conference for Emerging Technology (INCET)*, pages 1–4.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Chung-Wha (Chloe) Ki, Erin Cho, and Jung-Eun Lee. 2020. Can an intelligent personal assistant (ipa) be your friend? para-friendship development mechanism between ipas and their users. *Comput. Hum. Behav.*, 111:106412.

Evan King, Haoxiang Yu, Sangsu Lee, and Christine Julien. 2024. Sasha: Creative goal-oriented reasoning in smart homes with large language models. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 8(1).

Sunjun Kweon, Yeonsu Kwon, Seonhee Cho, Yohan Jo, and Edward Choi. 2023. Open-WikiTable : Dataset for open domain question answering with complex reasoning over table. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8285–8297, Toronto, Canada. Association for Computational Linguistics.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Qiwei Li, Zuchao Li, Ping Wang, Haojun Ai, and Hai Zhao. 2024. Hypergraph based understanding for document semantic entity recognition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2950–2960, Bangkok, Thailand. Association for Computational Linguistics.

Zeming Liu, Haifeng Wang, Zheng-Yu Niu, Hua Wu, Wanxiang Che, and Ting Liu. 2020. Towards conversational recommendation over multi-type dialogs. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1036–1049, Online. Association for Computational Linguistics.

Ewa Luger and Abigail Sellen. 2016. "like having a really bad pa": The gulf between user expectation and experience of conversational agents. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, page 5286–5297, New York, NY, USA. Association for Computing Machinery.

OpenAI, :, Ahmed El-Kishky, Alexander Wei, Andre Saraiva, Borys Minaiev, Daniel Selsam, David Dohan, Francis Song, Hunter Lightman, Ignasi Clavera, Jakub Pachocki, Jerry Tworek, Lorenz Kuhn, Lukasz Kaiser, Mark Chen, Max Schwarzer, Mostafa Rohaninejad, Nat McAleese, o3 contributors, Oleg Mürk, Rhythm Garg, Rui Shu, Szymon Sidor, Vineet Kosaraju, and Wenda Zhou. 2025. Competitive programming with large reasoning models. *Preprint*, arXiv:2502.06807.

Paul Jasmin Rani, Jason Bakthakumar, B. Praveen Kumaar, U. Praveen Kumaar, and Santhosh Kumar. 2017. Voice controlled home automation system using natural language processing (nlp) and internet of things (iot). In *2017 Third International Conference on Science Technology Engineering & Management (ICONSTEM)*, pages 368–373.

Dmitriy Rivkin, Francois Hogan, Amal Feriani, Abhisek Konar, Adam Sigal, Xue Liu, and Gregory Dudek. 2024. Aiot smart home via autonomous llm agents. *IEEE Internet of Things Journal*, pages 1–1.

Yingtian Shi, Xiaoyi Liu, Chun Yu, Tianao Yang, Cheng Gao, Chen Liang, and Yuanchun Shi. 2024a. Bridging the gap between natural user expression with complex automation programming in smart homes. *Preprint*, arXiv:2408.12687.

Zhengliang Shi, Shuo Zhang, Weiwei Sun, Shen Gao, Pengjie Ren, Zhumin Chen, and Zhaochun Ren. 2024b. Generate-then-ground in retrieval-augmented generation for multi-hop question answering. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7339–7353, Bangkok, Thailand. Association for Computational Linguistics.

John Sweller. 2011. Chapter two - cognitive load theory. volume 55 of *Psychology of Learning and Motivation*, pages 37–76. Academic Press.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, Nino Vieillard, Piotr Stanczyk, Sertan Girgin, and et al. 2024. Gemma 2: Improving open language models at a practical size. *Preprint*, arXiv:2408.00118.

Qwen Team. 2025. Qwq-32b: Embracing the power of reinforcement learning.

Hengyi Wang, Haizhou Shi, Shiwei Tan, Weiyi Qin, Wenyuan Wang, Tunyu Zhang, Akshay Nambi, Tanuja Ganu, and Hao Wang. 2024a. Multimodal needle in a haystack: Benchmarking long-context capability of multimodal large language models. *Preprint*, arXiv:2406.11230.

Hongru Wang, Rui Wang, Boyang Xue, Heming Xia, Jingtao Cao, Zeming Liu, Jeff Z. Pan, and Kam-Fai Wong. 2024b. AppBench: Planning of multiple APIs from various APPs for complex user instruction. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15322–15336, Miami, Florida, USA. Association for Computational Linguistics.

Huiming Wang, Liying Cheng, Wenxuan Zhang, De Wen Soh, and Lidong Bing. 2024c. Order-agnostic data augmentation for few-shot named entity recognition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7792–7807, Bangkok, Thailand. Association for Computational Linguistics.

Ke Wang, Jiahui Zhu, Minjie Ren, Zeming Liu, Shiwei Li, Zongye Zhang, Chenkai Zhang, Xiaoyu Wu, Qiqi Zhan, Qingjie Liu, and Yunhong Wang. 2024d. A survey on data synthesis and augmentation for large language models. *Preprint*, arXiv:2410.12896.

Zheng Wang, Shu Teo, Jieer Ouyang, Yongjun Xu, and Wei Shi. 2024e. M-RAG: Reinforcing large language model performance through retrieval-augmented generation with multiple partitions. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1966–1978, Bangkok, Thailand. Association for Computational Linguistics.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Ziqi Yin, Mingxin Zhang, and Daisuke Kawahara. 2024. Harmony: A home agent for responsive management and action optimization with a locally deployed large language model. *Preprint*, arXiv:2410.14252.

Haoxiang Yu, Jie Hua, and Christine Julien. 2021. Analysis of ifttt recipes to study how humans use internet-of-things (iot) devices. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, SenSys '21, page 537–541. ACM.

Xin Zeng, Xiaoyu Wang, Tengxiang Zhang, Chun Yu, Shengdong Zhao, and Yiqiang Chen. 2024. Gesturegpt: Toward zero-shot free-form hand gesture understanding with large language model agents. *Proc. ACM Hum.-Comput. Interact.*, 8(ISS).

Bowen Zhang, Kehua Chang, and Chunping Li. 2024a. Simple techniques for enhancing sentence embeddings in generative language models. In *Advanced Intelligent Computing Technology and Applications*, pages 52–64, Singapore. Springer Nature Singapore.

Wenxiao Zhang, Xiangrui Kong, Thomas Braunl, and Jin B. Hong. 2024b. Safeembodai: a safety framework for mobile robots in embodied ai systems. *Preprint*, arXiv:2409.01630.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Hao, Xu Han, Zhen Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. 2024c. ∞Bench: Extending long context evaluation beyond 100K tokens. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15262–15277, Bangkok, Thailand. Association for Computational Linguistics.

Zhuo Zhang, Xiangjing Hu, Jingyuan Zhang, Yating Zhang, Hui Wang, Lizhen Qu, and Zenglin Xu. 2023. FEDLEGAL: The first real-world federated learning benchmark for legal NLP. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3492–3507, Toronto, Canada. Association for Computational Linguistics.

Xiaoling Zhou, Wei Ye, Yidong Wang, Chaoya Jiang, Zhemg Lee, Rui Xie, and Shikun Zhang. 2024. Enhancing in-context learning via implicit demonstration augmentation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2810–2828, Bangkok, Thailand. Association for Computational Linguistics.

Zihao Zhu, Bingzhe Wu, Zhengyou Zhang, Lei Han, Qingshan Liu, and Baoyuan Wu. 2024. Earbench: Towards evaluating physical risk awareness for task planning of foundation model-based embodied ai agents. *Preprint*, arXiv:2408.04449.

A Appendix

Figure 6: ALL room types in HomeBench.

A.1 Data Collection

Table 6 presents the prompts we generated for user instructions using ChatGPT-3.5. To enhance the quality and accuracy of the generated instructions, we incorporated two example cases into the prompt. These examples provide a reference for the model, helping it better understand the task requirements and generate more precise and contextually appropriate responses to user instructions.

Figure 6 illustrates the multiple rooms with distinct functions that we designed. Each room's functionality has been carefully planned to ensure it aligns with realistic usage scenarios, covering a wide range of smart home applications. This structured design allows the model to handle diverse real-world contexts more effectively.

Figure 7 shows a virtual room construction example, we demonstrate how to select and configure smart home devices based on predefined rules to ensure consistency and logical compatibility. As shown in the code, certain devices in a living room setting have mutual exclusion constraints. For instance, an air conditioner, heater, and fan cannot coexist in the same space to prevent functional conflicts. Similarly, a humidifier and a dehumidifier are mutually exclusive to maintain proper environmental control. For other device categories, such as lighting, television, and audio systems, we randomly select devices from a predefined list and determine whether to add them to the current room. If the room already contains devices, new selections may be added as an expansion; if the room is empty, a new set of devices may be initialized, creating diverse configurations of virtual rooms. This construction logic not only ensures a reasonable selection of devices but also lays a foundation for further smart home scenario simulations and optimizations.

Table 7 lists all device categories included in our dataset, along with the corresponding methods that each category may contain. To enhance device diversity and ensure compatibility with the multifunctional nature of various smart home brands, we assigned a set of basic functions (e.g., turning on/off) to each device and randomly added additional functions. This approach allows devices to adapt dynamically to different situations, providing a more flexible and realistic user experience.

Figure 8 presents statistical insights into the distribution and types of devices across 100 virtual home environments. Within 12 functionally distinct rooms, users can interact with at least 47 operable devices spanning 10 different device categories. This comprehensive setup ensures a diverse and practical testing environment, effectively simulating real-world smart home scenarios.

A.1.1 Instruction Quality

In selecting the instruction generation method, we conducted a systematic comparative study. The experimental design encompassed two key dimensions. First, we compared the instruction generation capabilities of several large language models

Prompt

You are an artificial intelligence, skilled at converting machine instructions into commands issued by humans. Don’t be polite in tone, give instructions directly, and use a colloquial expression format. If multiple machine instructions are given, you need to convert them into one sentence. If the instruction contains an "explain" field, please note to generate human instructions of the amplitude adjustment type. Here are some examples:

Machine instructions: {’instruction’: ’set_intensity(30)’, ’device’: ’humidifier’, ’explain’: ’increase 2 percent’, ’room’: ’guest_bedroom’}

Human instructions: Increase the intensity of the humidifier by 2% in the guest bedroom.

Machine instructions: {’instruction’: ’set_temperature(26)’, ’device’: ’air_conditioner’, ’explain’: ’decrease 1 percent’, ’room’: ’guest_bedroom’}

Human instructions: Lower the air conditioner temperature in the guest bedroom by 1 degree.

Here are the machine instructions you want to modify:

Table 6: The prompt used to prompt LLM to generate the user instruction. In the prompt, we provide two examples to illustrate different adjustment methods more clearly. The first example demonstrates how to set a specific value directly for precise control of the target parameter. In contrast, the second example adopts a relative adjustment approach, modifying the value based on its current state to achieve dynamic tuning.

device type	method
LightDevice	turn_on;turn_off;set_brightness;set_color
AirConditionerDevice	turn_on;turn_off;set_temperature;set_mode;set_fan_speed;set_swing
HeatingDevice	turn_on;turn_off;set_temperature;set_mode;set_fan_speed
FanDevice	turn_on;turn_off;set_speed;set_swing
GarageDoorDevice	open;close
BlindsDevice	open;close
CurtainDevice	open;close;set_degree
AirPurifiersDevice	turn_on;turn_off;set_mode;set_fan_speed
WaterHeaterDevice	turn_on;turn_off;set_temperature;set_mode
MediaPlayerDevice	play;pause;stop;set_volume;set_song;set_artist;set_style
VacuumRobotrDevice	start,pause;stop;charge;set_mode;set_cleaning_area
AromatherapyDevice	turn_on;turn_off;set_intensity;set_interval
TrashDevice	pack
HumidifierDevice	turn_on;turn_off;set_intensity;set_mode
DehumidifiersDevice	turn_on;turn_off;set_intensity;set_mode

Table 7: All device types and their methods in HomeBench.

```python
class VisualLivingRoom:
    def __init__(self) -> None:
        self.name = "living_room"
        self.devices = []
        self.unexist_devices = []
        self.devices.append(random.choice(LightDeviceList)("on"))
        if random.random() > 0.5:
            self.devices.append(random.choice(AirConditionerDeviceList)("on"))
            self.unexist_devices.append(random.choice(HeatingDeviceList)("on"))
            self.unexist_devices.append(random.choice(FanDeviceList)("on"))
        else:
            self.devices.append(random.choice(HeatingDeviceList)("on"))
            self.devices.append(random.choice(FanDeviceList)("on"))
            self.unexist_devices.append(random.choice(AirConditionerDeviceList)("on"))
        random.choice([self.devices, self.unexist_devices]).append(random.choice(CurtainDeviceList)("on"))
        random.choice([self.devices, self.unexist_devices]).append(random.choice(AirPurifiersDeviceList)("on"))
        if random.random() > 0.5:
            self.devices.append(random.choice(HumidifierDeviceList)("on"))
            self.unexist_devices.append(random.choice(DehumidifiersDeviceList)("on"))
        else:
            self.devices.append(random.choice(DehumidifiersDeviceList)("on"))
            self.unexist_devices.append(random.choice(HumidifierDeviceList)("on"))
        random.choice([self.devices, self.unexist_devices]).append(random.choice(AromatherapyDeviceList)("on"))
        random.choice([self.devices, self.unexist_devices]).append(random.choice(TrashDeviceList)("on"))
        random.choice([self.devices, self.unexist_devices]).append(random.choice(MediaPlayerDeviceList)("play"))
        random.choice([self.devices, self.unexist_devices]).append(random.choice(PetFeederDeviceList)("on"))
        self.random_initialize()
        self.state = self.get_status()
```

Figure 7: Smart home virtual room configuration: ensuring logical device selection.

Prompt

You are ’AI’, a helpful AI Assistant that controls the devices in a house. Complete the following task as instructed or answer the following question with the information provided only. The current status of the device and the methods it possesses are provided below, please only use the methods provided. Output ërror_inputẅhen operating non-existent attributes and devices. Only output machine instructions and enclose them in { }.

<home_state>

The following provides the status of all devices in each room of the current household, the adjustable attributes of each device, and the threshold values for adjustable attributes:

......(Home device status, Table 9 shows an example.)

</home_state>

<device_method>

The following provides the methods to control each device in the current household:

......(Device methods, Table 10 show an example.)

</device_method>

——————————————-

Here are the user instructions you need to reply to.

<User instructions:>

The user instructions.

<Machine instructions:>

Table 8: The prompt used to prompt LLM to generate an operation. Blue is home state and orange is device method.

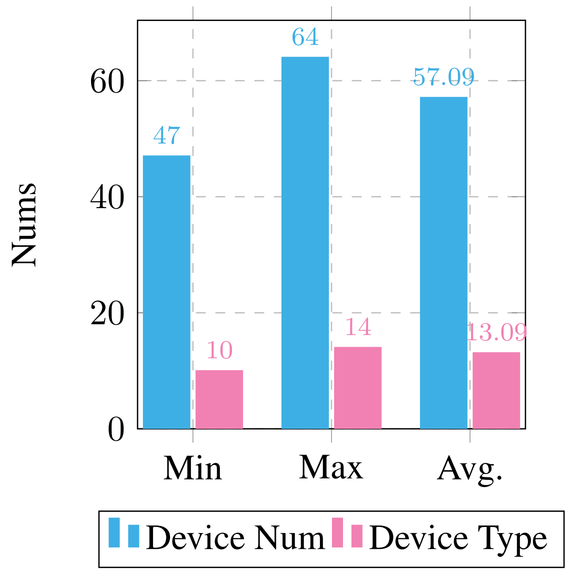

Figure 8: The statistics of the number and types of devices in 100 home environments.

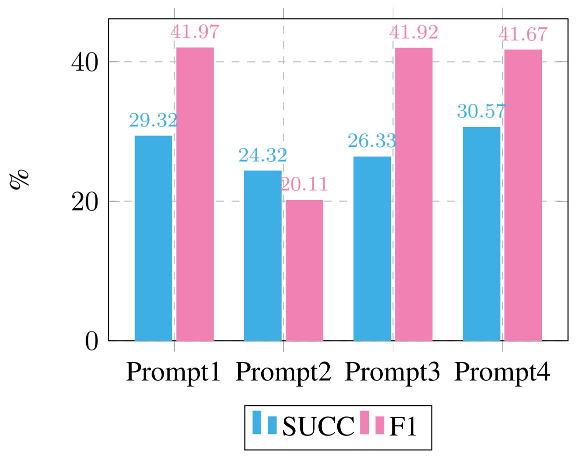

Figure 9: Performance of Gemma2-9B-Instruct on test dataset under different prompts. Prompt 1 and Prompt 2 were manually written. Prompt 3 was generated by modifying Prompt 1 using GPT-4. Prompt 4 was also manually written and is the final prompt we used.

Example

master_bedroom:
—light
——-state: off
——-brightness: 35 (range: 0 - 100)
——-color: [232, 50, 198]
–heating
——-state: off
——-mode: fan_only (options['heat', 'fan_only'])
——-fan_speed: auto (options['auto', 'low', 'medium', 'high'])
–fan
——state: on
——swing: auto (options['auto', 'up', 'middle', 'down'])
....

Table 9: Examples of home device status.

Example

master_bedroom.light.turn_on();
master_bedroom.light.turn_off();
master_bedroom.light.set_brightness(brightness:int);
master_bedroom.light.set_color(color:typing.Tuple[int, int, int]);
master_bedroom.heating.turn_on();
master_bedroom.heating.turn_off();
master_bedroom.heating.set_mode(mode:str);
master_bedroom.heating.set_fan_speed(fan_speed:str);
master_bedroom.fan.turn_on();
master_bedroom.fan.turn_off();
master_bedroom.fan.set_swing(swing:str);

Table 10: Examples of home device methods.

Date Type	User Instruction	Output
VS	Set the brightness of the light on the balcony to 50.	balcony.light.set_brightness(50)
IS	Pause the media player in the living room.	error_input
VM	Lower the air conditioner temperature in the guest bedroom to 20 degrees, set the brightness of the light in the foyer to 50, and increase the intensity of the dehumidifier in the study room by 32%.	guest_bedroom.air_conditioner.set_temperature(20), foyer.light.set_brightness(50), study_room.dehumidifiers.set_intensity(70)
IM	Set the intensity of the humidifier to 60 in the study room, adjust the volume of the media player to 60 on the balcony, and set the degree of the curtain to 20 on the balcony.	error_input, error_input, error_input,
MM	Set the air conditioner temperature to 16 degrees in the guest bedroom, turn off the lights in the study room, decrease the media player volume by 30 percent on the balcony, set the dehumidifier intensity to 60 in the master bedroom, and adjust the heating temperature to 27 degrees in the bathroom.	guest_bedroom.air_conditioner.set_temperature(16), study_room.light.turn_off(), error_input, error_input, bathroom.heating.set_temperature(27)

Table 11: Examples of different data types in HomeBench.

(including ChatGPT-3.5, GPT-4, and Qwen-72B). Second, we established a benchmark control group comprising human-written instructions. For each method, we generated 200 instruction samples and performed a comprehensive evaluation based on three critical metrics: generation quality, generation efficiency, and economic cost.

The experimental results revealed that in terms of generation quality, ChatGPT-3.5 performed comparably to human-written instructions. However, it demonstrated significant advantages in both efficiency and cost-effectiveness. Specifically, ChatGPT-3.5 required an average of only 2 minutes to generate instructions at a cost of less than 0.5 dollars, whereas manual instruction writing took nearly 4 hours with an approximate cost of $25.

Based on these findings—comparable quality but substantially superior efficiency and cost—we ultimately selected ChatGPT-3.5 as our instruction generation tool.

A.2 Experiment

Implementation Details. Our experiments were conducted using a combination of open-source models, closed-source models, and fine-tuned models, leveraging different hardware and APIs as appropriate. For open-source models, all experiments were performed on NVIDIA RTX 3090 GPUs, ensuring efficient execution and evaluation. For closed-source models and Qwen2.5-72B-Instruct, we utilized APIs provided by OpenAI, enabling us to benchmark their performance under the same experimental conditions. Fine-tuning experiments were conducted on NVIDIA A800 GPUs, leveraging their high memory capacity and computational power to optimize model performance. We employed Hugging Face's PEFT (Parameter-Efficient Fine-Tuning) framework to fine-tune the models efficiently while minimizing computational costs. As shown in the figure 10, we use LoRA to fine-tune the Qwen model.

Table 8 presents the prompt designs used in all model tests. Each prompt contains information about the status of room devices to ensure that the model accurately understands the current environment and generates appropriate responses (see Table 9). Additionally, we have listed the invocation methods for all devices to enable the model to execute relevant operations correctly, as detailed in Table 10.

To determine the most suitable prompt, we conducted experiments with four different prompts.

```
lora_config = LoraConfig(
    r=16,
    lora_alpha=32,
    target_modules=["q_proj", "k_proj", "v_proj", "o_proj", "gate_proj", "up_proj", "down_proj"],
    lora_dropout=0.1,
    bias="none",
    task_type="CAUSAL_LM"
)
```

Figure 10: The LoRA configuration used for fine-tuning the Qwen2.5-7B-Instruct model.

Prompt 1 and Prompt 2 were manually written to explore the effectiveness of human-designed prompts. Prompt 3 was derived from Prompt 1 but refined using GPT-4 to evaluate the capability of large language models in prompt optimization. Prompt 4 was also manually written and was ultimately selected as the final prompt version. Figure 9 illustrates the performance of different prompts on the test dataset, comparing their effectiveness and providing insights into the selection of the optimal prompt.

A.2.1 Few Shot

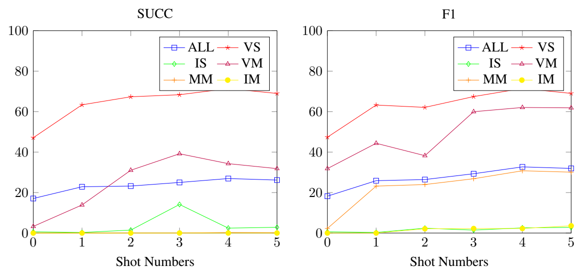

Figure 11: The performance of Llama3-8B-Instruct model in adding different types of data samples (the order of adding is: VS, IS, VM, MM, IM).

Table 12 presents the different categories of samples used to evaluate various few-shot learning methods. These samples cover a range of device function categories, listed from top to bottom: VS, IS, VM, MM, and ME. By incorporating these diverse categories, we can more effectively assess the model's performance across different real-world smart home scenarios.

Few shot

Here are a few examples, your output format should be consistent with the results provided in the example:

<example1>

<User instructions:> Lower the air conditioner temperature in the guest bedroom by 1 degree.

<Machine instructions:> {guest_bedroom.air_conditione.set_temperature(26)}

</example1>

<example2>

<User instructions:> Turn on the light in the living room.

<Machine instructions:> {error_input} (If there is no device or if there is no attribute or method that can be operated on the device, please output as "error_input".)

</example2>

<example3>

<User instructions:> Set the intensity of the humidifier to 20% in the guest bedroom, adjust the intensity of the dehumidifiers by increasing it by 30% in the study room, and set the intensity of the dehumidifiers to 40% in the study room.

<Machine instructions:>{guest_bedroom.humidifier.set_intensity(20),study_room.dehumidifiers.set_intensity(10),study_room.dehumidifiers.set_intensity(40),}

</example3>

<example4>

<User instructions:> Increase the intensity of the aromatherapy in the corridor by 4%, set the brightness of the lights in the garage to 40, adjust the fan speed of the air conditioner in the living room to low, and set the volume of the media player in the living room to 0.

<Machine instructions:> {corridor.aromatherapy.set_intensity(30),error_input,error_input,living_room.media_player.set_volume(0),}

</example4>

<example5>

<User instructions:> Turn on the air purifiers in the garage and increase the volume of the media player in the living room by 50 percent.

<Machine instructions:> {error_input,error_input}

</example5>

Table 12: Different types of few-shot, the sequence from top to bottom is VS, IS, VM, MM, IM.

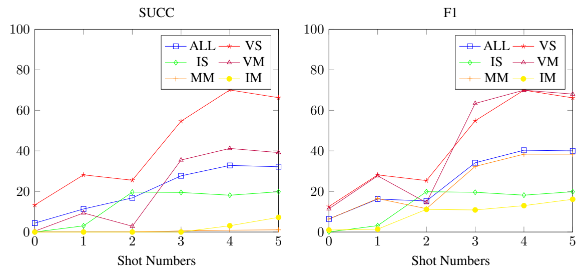

Figure 12: The performance of Gemma2-9B-Instruct model in adding different types of data samples (the order of adding is: VS, IS, VM, MM, IM).

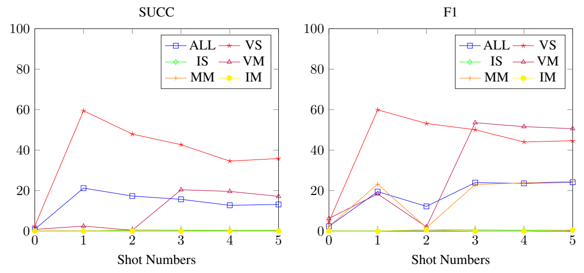

Figure 13: The performance of Mistral-7B-v0.3 model in adding different types of data samples (the order of adding is: VS, IS, VM, MM, IM).

Figures 11, 12, and 13 illustrate the performance of Llama3-8B-Instruct, Gemma2-9B-Instruct, and Mistral-7B-v0.3, respectively, under different few-shot settings. The results from Llama3-8B-Instruct and Gemma2-9B-Instruct align with our findings in Section 5.1: as the number of few-shot examples increases, the models generally exhibit improved performance, though the degree of enhancement varies across different data types. However, we also observed a performance decline in certain tasks, particularly IS and VS, as the number of shots increased. This decline is likely due to the lack of home device state information in the additional shots. Since the added examples only include user instructions and device operations, the model struggles to accurately determine device status, leading to confusion and a drop in performance.

Interestingly, Mistral-7B-v0.3 displays a unique behavior: it reaches its performance ceiling at just 2-shot, with no significant improvement beyond this point. This could be attributed to the fact that we did not use an instruction-tuned version of Mistral, making it less capable of fully understanding the specific requirements of our task. This also explains Mistral's relatively poor performance in the main experiments, further reinforcing our hypothesis.

A.2.2 RAG Setting and Case Study

We segmented device states and methods based on room categories and input these segments into the LLM to extract hidden states from the final layer. This approach allowed us to capture meaningful representations of device functionality and contextual dependencies.

Following the methodology outlined in (Zhang et al., 2024a), we incorporated the phrase "After thinking step by step" into the prompt. This addition helps the model better process sequential reasoning and compress semantic information effectively, improving its understanding of device interactions. After obtaining embeddings for both the segmented chunks and user instructions, we computed cosine similarity to facilitate retrieval. The retrieval process follows these conditions:

- If the similarity score exceeds 0.5 and the number of retrieved chunks is greater than 3, we return the results directly.
- If fewer than 3 chunks are retrieved, we return the top three chunks with the highest similarity scores to ensure the model has sufficient contextual information for accurate execution.

This strategy enhances retrieval efficiency and ensures that the model has a reliable and relevant device context for decision-making.

In the Table 13, we provide case examples of context retrieval errors across different data categories. This context mismatch leads to a counterintuitive phenomenon—performance degradation on simple and effective tasks, but unexpected improvement on more challenging ones. Upon further analysis, we find that the root cause lies in insufficient embedding similarity between device states/methods and user instructions, which hampers the similarity-based retrieval mechanism's ability to effectively distinguish between different context fragments. To address this issue, we propose training a specialized retrieval optimization model to enhance context alignment accuracy.

A.2.3 Error Analysis

Table 14 provides a detailed breakdown of the proportion and quantity of different error types across various data types. It presents the distribution of Unfaithfulness, In-context Attention Error, and Key Error for each data type, along with their respective percentages and absolute counts. This allows for a more intuitive analysis of the error patterns and

Data Type	Input	Golden	Genrated
VS	Set the brightness of the light on the balcony to 50. balcony.light.set_brightness(50)	balcony.light.set_brightness(50)	error_input
IS	Pause the media player in the living room.	error_input	error_input
VM	Lower the air conditioner temperature in the guest bedroom to 20 degrees, set the brightness of the light in the foyer to 50, and increase the intensity of the dehumidifier in the study room by 32%.	guest_bedroom.air_conditioner.set_temperature(20), foyer.light.set_brightness(50), study_room.dehumidifiers.set_intensity(70)	error_input, error_input, error_input,
IM	Set the intensity of the humidifier to 60 in the study room, adjust the volume of the media player to 60 on the balcony, and set the degree of the curtain to 20 on the balcony.	error_input, error_input, error_input,	error_input, error_input, error_input,
MM	Set the air conditioner temperature to 16 degrees in the guest bedroom, turn off the lights in the study room, decrease the media player volume by 30 percent on the balcony, set the dehumidifier intensity to 60 in the master bedroom, and adjust the heating temperature to 27 degrees in the bathroom.	guest_bedroom.air_conditioner.set_temperature(16), study_room.light.turn_off(), error_input, error_input, bathroom.heating.set_temperature(27)	error_input, error_input, error_input, error_input, error_input,

Table 13: Model output when there no useful information in the context. Red indicates an incorrect output, while green represents a correct output.

Type	Unfaithfulness	In-context Attention Error	Key Error
VS	50.53/805	7.53/120	41.93/668
IS	-	100/873	-
VM	53.83/71	60.90/81	44.36/59
IM	-	100/37	-
MM	23.45/732	44.44/1387	41.15/1253

Table 14: The proportion of different error types for different data types. The former is the ratio, and the latter is the quantity.

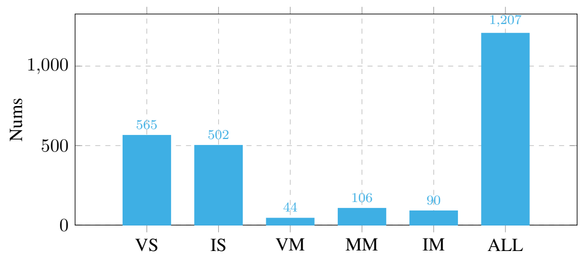

Figure 14: OOD data distribution.

potential issues affecting the model's performance across different data types.

A.2.4 OOD

We have created two new smart devices: beds and pet feeders to construct ood data. The data distribution is shown in the Figure 14.

A.2.5 Inference Latency

In Table 15, our tests conducted on an RTX 3090 platform demonstrate that the inference latency of the locally deployed model can be kept within a few seconds, which generally satisfies practical requirements. However, to further improve the user experience—ideally by reducing latency to under one second—enhancing inference speed remains a key focus of our future work.

Models	LLaMa3-8b	Qwen2.5-7B	Mistral-7B	Gemma2-9B	Deepseek-V3	GPT-4o
Latecy	2.49s	1.99s	3.05s	6.92s	6.79s	1.16s

Table 15: Inferface latecy