Article | Received 25 December 2024; Accepted 19 February 2025; Published 4 March 2025
https://doi.org/10.55092/sc20250004

A general AI agent framework for smart buildings based on large language models and ReAct strategy

Xiangjun Yan[1], Xincong Yang[1,2,*], Nan Jin[3], Yu Chen[1] and Jiaqi Li[1]

1 School of Civil and Environmental Engineering, Harbin Institute of Technology, Shenzhen, China
2 Guangdong Provincial Key Laboratory of Intelligent and Resilient Structures for Civil Engineering, Harbin Institute of Technology, Shenzhen, China
3 Safe Urban Development Institute of Science and Technology (Shenzhen), Shenzhen, China

* Correspondence author; E-mail: yangxincong@hit.edu.cn.

Highlights:

- A general AI agent framework is proposed for indoor human-computer interactions.
- The ReAct strategy integrates the LLMs' capacities in reasoning and action planning.
- The proposed framework achieves high success rate in virtual building simulations.
- The LLM-powered agent can be integrated with existing IoT systems seamlessly.

Abstract: Smart buildings represent a significant trend in the future of the construction industry. The performance of human-computer interaction plays a vital role in achieving this from a human perspective. However, existing human-computer interaction algorithms are often limited to simple commands and fail to meet the complex and diverse needs of users. To address this issue, this paper introduces large language models (LLMs) and AI agents into smart buildings, proposing a general AI agent framework based on the ReAct strategy. The LLM serves as the system's brain, responsible for reasoning and action planning, while tool calling mechanism puts the LLM's plans into practice. Through this framework, developers can rely on prompt engineering alone to enable the LLM to interpret user intent accurately, perform appropriate actions, and manage conversation history effectively, without any pre-training or fine-tuning. To examine this framework, an experiment was conducted in a virtual building, which showed that the proposed agent successfully completed 91% of simulated tasks. Additionally, the agent was deployed on a single-board computer to control devices in a model building, demonstrating its effectiveness in the real world. The successful operation of the agent in this environment highlighted the potential applications of the proposed framework using existing IoT systems, providing a new perspective for the upgrading of human-computer interaction systems in smart buildings in the near future.

Keywords: smart building; human-computer interaction; large language model; AI agent; reasoning and acting; Internet of Things

1. Introduction

Smart buildings are dynamic systems, which can sense indoor and outdoor states, allow occupants to interact with their components, and create safe, comfortable, and efficient environments for working and living by responding to people's needs promptly. They also make positive contributions to cost reduction, resource conservation, and energy savings [1]. In recent years, smart buildings have garnered increasing attention. Taking smart homes in residential buildings as an example, Juniper Research reports that by 2025, approximately 13.5 billion home automation systems will be frequently used [2]. It is foreseeable that smart buildings will become an indispensable infrastructure in future society.

The integration of cutting-edge technologies is the foundation of smart buildings. This integration encompasses a wide array of technologies, such as sensors, cloud and fog computing, software engineering, and human-computer interaction algorithms [3]. As a result of the advancements in embedded development, an IoT system covering an entire building can now be established. However, when it comes to human-computer interaction, despite the emergence of voice recognition, gesture recognition, and other methods, there remains a lack of deep understanding of user intent. For example, when users give implicit and tacit commands like "I want to be alone for a while; I don't want to be disturbed." or "Set the light mode to the same as yesterday," the system may not function as expected.

The advent of large language models has enabled machines to understand complex human language, while the concept of AI agents has bridged the gap between LLMs and the physical world, making it possible to build LLM-driven human-computer interaction systems. In this paper, we design an AI agent framework for smart buildings based on the ReAct strategy [4], which consists of four main modules: model hub, prompt template, tool manager and memory manager, along with an auxiliary output parser program. Through close collaboration among these components, the framework achieves the desired functionality. We conducted two experiments with a virtual building in Python code and a wooden model building in the real world to validate the effectiveness of the proposed framework.

2. Related work

This chapter reviews the research on human-computer interaction in smart buildings and the application of LLMs in the Architecture, Engineering, and Construction (AEC) domain to find the shortcomings of existing research, identify research gaps and justify the necessity of this study.

2.1. Human-computer interaction in smart buildings

Owing to the development of information technology, the interaction approaches between residents and their furniture and appliances is getting increasingly diverse.

Using a remote control to send signals is a traditional but common method of human-computer interaction, but this requires matching receivers on the devices. To overcome this limitation, Rashid *et al.* [5] designed a universal hand-held clicker with ultra-wideband (UWB) localization tag and inertial measurement unit (IMU), which can obtain the user's position and orientation, determine the target device with the support of a building information model (BIM), and then control the device through an indoor IoT network.

Some researchers attempted to employ gestures or postures as media for human-computer interaction in buildings. Vogiatzidakis *et al.* [6] proposed a method for interacting with household appliances using two-handed gestures, with one hand specifying the device and the other indicating control commands. Microsoft Kinect, a motion capture device, is used for gesture recognition. Park *et al.* [7] developed a user authentication system by analyzing Doppler signals to identify hand movements and combining them with virtual buttons in the air. Wang *et al.* [8] integrated a Convolutional Neural Network-based posture recognition algorithm into the smart building control system, enabling the building to evaluate people's thermal and visual comfort by observing their behavior and adjust the Heating, Ventilation, and Air Conditioning (HVAC) system and sun visors accordingly.

Eye movements and blinks are also ways to convey information for humans. Some researchers have developed human-computer interaction systems based on eye movements. Klaib *et al.* [9] used multiple cameras for eye tracking to determine the user's gaze position. This approach was combined with Azure Cloud's voice interaction service to control devices. Zhang *et al.* [10] proposed to detect the user's conscious blinking behavior by electrooculography and control home appliances with a display showing various icons, providing convenience for severe spinal cord injury patients.

The aforementioned interaction methods inherently possess limited expressive capabilities, making it challenging to convey complex user instructions. Instead, human language is rich and diverse, offering a wide range of expressions with various meanings, emotions and intentions. Besides, direct natural language communication with computers is a straightforward and efficient interaction method, making it a research hotspot. The current challenge in voice interaction lies in the difficulty for computers to accurately recognize and interpret the users' instructions. In Yang's [11] smart home system, voice control is achieved by matching the user's voice with preset voice commands in a voice library. Tiwari *et al.* [12] enhanced the home voice assistant with speaker recognition using a Gaussian Mixture Model, enabling it to process personalized commands such as "Switch on my TV." The user's voice is converted to text via cloud services, and the assistant extracts device, operation and other information from the text based on predefined keywords. Peña-Cáceres *et al.* [13] trained Naive Bayes and Random Forest models for intent recognition, which can analyze the input sentences to determine the target device, room and desired state. However, these models only consider word frequency in a Bag of Words representation, ignoring word order and semantic information. Stefanenko *et al.* [14] trained a neural network with two hidden layers, where the input is a Bag of Words representation derived from user commands, and the output is the position of the virtual toggle switch used for robot control. This system excels at handling some ambiguous terms in users' instructions, such as "faster" or "louder". Shin *et al.* [15] proposed a framework for retrieving information from BIM through voice interaction. The framework first converts speech into text and then transforms the text into SQL-compliant query statements to extract information from database. Desot *et al.* [16] utilized the ESPnet model to analyze raw voice signals and output device and operation information, avoiding errors caused by voice-to-text conversion. Linares-Garcia *et al.* [17] developed a voice assistant based on Google Actions Builder to answer the workers' questions. Although this study focuses on the construction stage of buildings rather than the operational stage, it still provides valuable insights for voice interaction in smart buildings.

Although existing research has extensively explored interaction forms in buildings and many kinds of algorithms have been proposed, including rule-based methods, traditional machine learning methods and classical deep learning methods, there are still some shortcomings. Rule-based methods can only handle simple, direct and explicit user commands. The latter two methods possess the capability of learning, but they can only make limited effects because of insufficient model scale and training data, leaving them far from achieving genuine intelligence that meets user needs. Additionally, the models in these studies are specialized models. When the scenario changes, such as adding some new devices like water heaters or increasing the number of doors from 5 to 10, the original models are no longer applicable and need modification and retraining, which hinders the promotion and application of smart building systems.

2.2. Application of LLMs in the AEC domain

The release of the Transformer architecture in 2017 greatly changed the landscape of deep learning and marked a significant milestone in the history of artificial intelligence. Supported by this architecture, many pre-trained large language models (LLMs) have emerged. Gradually, LLMs have become increasingly accurate and comprehensive in understanding user needs. They also demonstrate outstanding few-shot generalization capabilities [18]. To date, although general artificial intelligence is still far from us, LLMs have shown a certain degree of general intelligence behavior in NLP tasks. They are expected to make a difference to various industries and become new intelligent assistants for human beings. In the AEC domain, some scholars have already noticed the potential of LLMs and explored their usage in solving some problems which were intractable for computers previously.

Xu *et al.* [19] constructed a domain knowledge database of underground engineering, and fine-tuned ChatGLM3 to create a more specialized question-and-answer robot. Pu *et al.* [20] fine-tuned MiniGPT4 with text and image data from construction sites to generate construction inspection reports automatically. Amer *et al.* [21] fine-tuned the GPT-2 model for drafting construction plans. Hussain *et al.* [22] utilized ChatGPT to generate dialogue for virtual instructors in a VR construction system, aimed at enhancing the safety education for workers. Zhang *et al.* [23] fed building operational data into GPT-3.5 and GPT-4 in order to automatically identify energy waste problems in building HVAC systems. Ahmadi *et al.* [24] employed multiple LLMs to analyze construction accident reports, extracting key information such as accident time and injury cause. Ghasemi *et al.* [25] applied GPT-4 to address cost estimating and bid pricing tasks of construction projects. Wong *et al.* [26] proposed using LLMs to automatically identify risks in contract clauses of construction projects, thereby reducing potential losses for project stakeholders.

However, the application of LLMs in the AEC domain is mostly limited to chatbot phase, which means "user inputs text and AI outputs text." This mode restricts the AI within a closed space defined by its inherent knowledge, additional domain knowledge and user prompts, preventing it from interacting with the broader external world or being used to construct intelligent building systems. Indeed, generative LLMs are merely tools for completing text based on preceding content and outputting text is their sole responsibility. However, if we inform LLMs of how to use tools for interacting with the external world, they can specify which tools are needed to fulfil the current task. In this condition, if we require LLMs to follow strict grammatical rules and output code or structured text, and then use a rule-based program to parse the output and execute the actual tool calling, the interaction between LLMs and the external world can be indirectly achieved. This system is known as an LLM based autonomous agent, where the LLM plays a role as the central controller, and the tools extend its capabilities [27]. OpenAI's WebGPT [28] is an early practical implementation of this concept. On the other hand, the introduction of chain-of-thought (CoT) prompting inspires the potential of LLMs in performing multi-step reasoning through intermediate language steps [29]. ReAct agent [4] successfully integrates the reasoning capacity and action planning capacity of LLMs, which inherently incorporates CoT method and tool calling mechanism, forming a more comprehensive AI agent development pattern.

Existing research has demonstrated the feasibility of using AI agents to control robots [30]. Undoubtedly, it highlights the immense potential of applying LLMs to smart building systems. However, there is still a lack of such practice in the AEC domain, and further research is needed to fill this research gap.

3. Proposed method

To construct an AI agent for smart building scenarios, this paper proposes a framework as illustrated in Figure 1 based on the ReAct strategy. Model hub, prompt template, tool manager and memory manager are the four main modules of the agent. Additionally, an output parser program is included to interpret the LLMs' output and determine data flow directions.

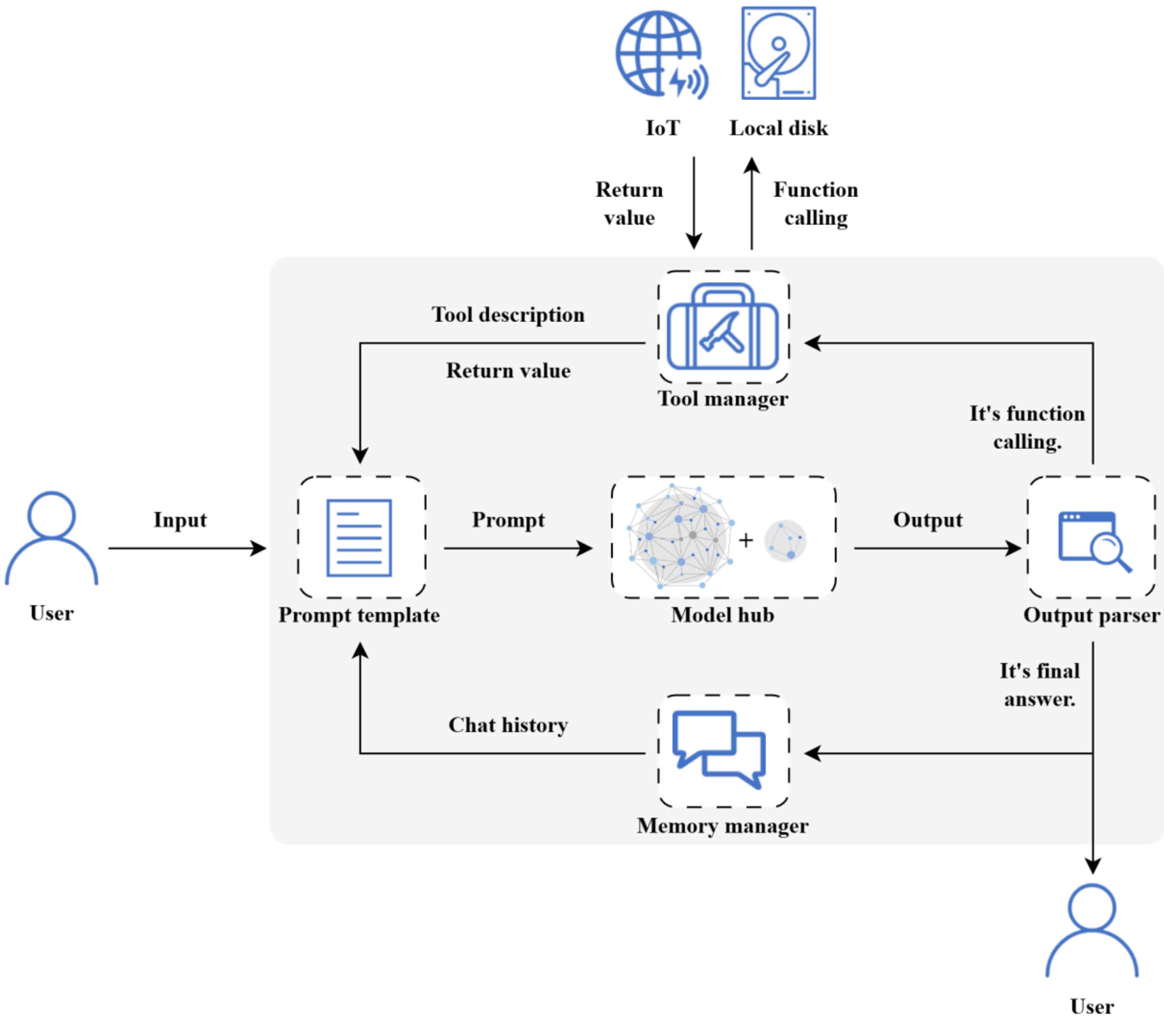

Figure 1. Framework of the proposed agent.

Upon receiving user's input, the user's instructions, tool descriptions and chat history are filled into the prompt template to form a prompt, which is input into the model hub subsequently. The LLM in the model hub considers whether tools are needed to complete the current task and generates different outputs based on its thought. After that, the output parser analyzes the string output from the model hub and searches for content related to the final answer within it. If no such content is found, it indicates that the LLM believes tool calling is necessary, and the system proceeds into the tool manager, which calls the corresponding function based on the function name and parameters listed by the LLM and obtains its return value, thus enabling interaction between the agent and the external IoT system or data in the local disk. After the function execution is finished, the tool manager updates the prompt based on the function's return value and a new round of circulation is initiated. If the output parser finds content related to the final answer, it indicates that the LLM believes no tools are needed and the final answer can be directly generated. Then, the final answer is sent to the user and stored by the memory manager as the chat history for future conversations.

3.1. Model hub

LLM is the core of the agent, responsible for identifying user intentions and writing out the thought process and action plan in text form, directly influencing the agent's operational direction. After deliberation, the LLM should accurately outline the idea to solving the user's problem and list a function remaining to be called next and the input parameters of it. If any anomaly is detected in the return value of previous functions, it is also necessary for the LLM to recognize the issue in time and re-plan its action.

Currently, some LLMs have tens of billions or even more parameters, the scale of their training data also needs to be measured in terabytes (TB). The vast amount of parameters and training data have grant these models excellent comprehension, reasoning, and expression capabilities. However, the drawbacks of such complex models are also evident: they consume significant computational resources, respond at a low speed and incur high costs. On the other hand, some manufacturers have also launched lightweight LLMs with several billion parameters, but they have been pre-trained on large datasets prior to applications. These models can also handle some conversational tasks and have the advantages of faster computing speed and low usage costs. Therefore, this paper adopts a model hub that combines both complex LLM and lightweight LLM. As shown in Figure 2, the complex LLM acts as a multifunctional advanced AI assistant, understanding the prompt globally and drafting a plan for the task, while the lightweight LLM focuses on a specific aspect, acting as an AI programmer which is responsible for converting the plan into code and providing it to the output parser, thereby bridging the gap between thought and action.

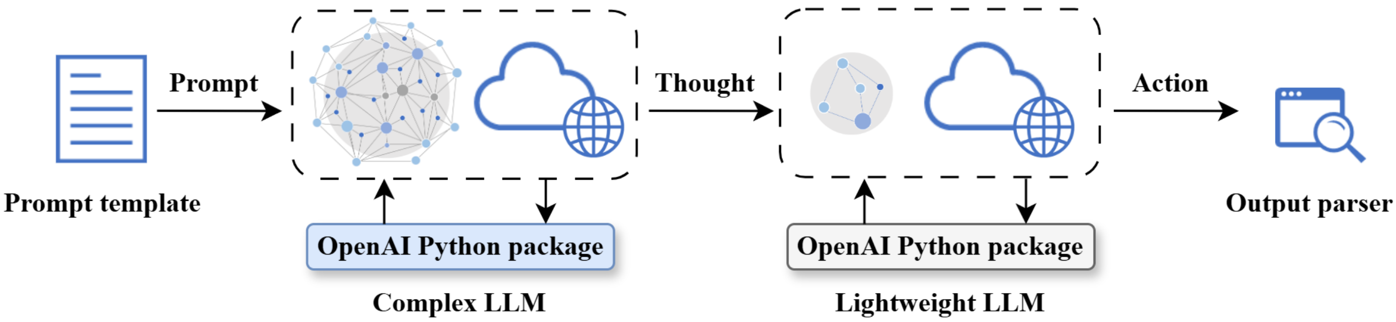

Figure 2. Model hub.

To avoid excessive encapsulation and maintain the clarity and transparency of the agent's operational flow, we use Python's native OpenAI package to call cloud-based LLMs with OpenAI-compatible API via HTTPS.

3.2. Prompt template

When using LLMs for text generation, it is necessary to provide some initial text as the starting point of the generation. The initial text is known as "prompt". The prompt is the only way for an LLM to obtain external information and largely determines the quality of subsequent generated content. We designed the prompt template based on the ideas of react-chat [31] and react-json [32]. The prompt template for the complex LLM, as shown in Table 1, includes six parts: "background," "role," "tool description," "format constraints," "chat history," and "new input." The necessary information will be inserted into the placeholders in it and form a resultant prompt which is named "Prompt A".

Table 1. The prompt template of complex LLM.

Category	Content
background	Current Time: {time} Current Location: {location}
role	You are a helpful AI assistant. Assistant is designed to be able to assist with a wide range of tasks, from answering simple questions to providing in-depth explanations and discussions on a wide range of topics. As a language model, Assistant is able to generate human-like text based on the input it receives, allowing it to engage in natural-sounding conversations and provide responses that are coherent and relevant to the topic at hand.
tool description	You have access to the following tools: {tools}
format constraints	You MUST use the following format to respond: Thought: Describe the problem you need to solve next, and think about whether you need to use tools. If you think you can respond directly to the user without using a tool, please reply: Final Answer: string \ Put your final response here. If you think you need to use a tool, please reply: Action: A JSON object that contains the tool's name and the tool's inputs. Observation: the result of action Thought: a new round of thinking ... Thought/Action/Observation can repeat several times until you think you no longer need any tool. At this point, please reply: Thought: I now know the final answer. Final Answer: string \ Put your final response here. If you do not reply in this format, you may cause a programming error.
chat history	The chat history between the user and the AI: {chat_history}
new input	The user's new input: Human: {input} The AI program is running now. Here is the process:

The “background” section provides the information on the current time and location, helping the LLM to provide correct inputs when calling tools related to time and location. On the other hand, once the training of an LLM is completed, the LLM’s inherent knowledge will no longer be updated. Over time, its answers will gradually lose timeliness. Besides, the LLM itself also lacks the ability to obtain the current geographical location. Therefore, filling time and location information into the prompt can also help the model to recognize the boundaries of its capabilities and reduce the occurrence of hallucinations.

The “role” section defines the role, responsibilities and personality characteristics of the LLM in the dialogue. As a human-like AI, after receiving this instruction, the LLM will make its output as consistent as possible with the above requirements to better fulfill the tasks delivered by the user.

The “tool description” section lists the tools which are accessible to the agent. For each tool, it should include the function name, input parameters and their types, the function purpose, and other necessary usage instructions. During the agent’s operation, this section is maintained by the tool manager and automatically filled into the {tool} placeholder.

The “format constraints” section stipulates the format that the LLM should use in its responses so that the output parser can process it conveniently. The LLM can handle natural language, but ordinary Python programs do not have this ability. However, the output of the LLM ultimately needs to be connected to ordinary Python programs, so pre-defining the output format is necessary. If the response is semantically correct but its format is incorrect, the output parser may fail to extract key information within it, which will lead to program errors.

The “chat history” section records the previous conversations between the user and the AI, enabling multi-round dialogue functionality. For example, users can continue to ask questions about a topic which was discussed with the AI previously. This section is maintained by the memory manager.

The “new input” section is the last part of the prompt template, where the user’s instructions in the current round of dialogue are placed.

Since the lightweight LLM is only responsible for writing JSON, its prompt template is more concise, as shown in Table 2. The resultant prompt derived from it is named “Prompt B”.

Table 2. The prompt template of lightweight LLM.

Category	Content
background	Current Time: {time} Current Location: {location}
role	You are a helpful AI programmer. You’re good at converting the input to JSON.
tool description	You have access to the following tools: {tools}
format constraints	You MUST use the following format to respond: ```json {{ "action": string, \ The name of the tool to be used. "action_inputs": list \ The inputs for the tool, represented as a list. For boolean values, use true and false instead of True and False. }} ``` Only one JSON should be included in your response. If you do not reply in this format, you may cause a programming error.

After the conversation starts, Prompt A is first input to the complex LLM, which continues to write the Thought section. For simple daily conversations that do not require tools, the LLM directly generates the Final Answer after completing the Thought section, ending the current round of dialogue. The Final Answer is presented to the user and stored in the memory manager. If tools are needed to fulfill the user's task, the generated Thought is appended to the end of Prompt B to form a new Prompt B as the input for the lightweight LLM which writes the Action section in JSON format. Subsequently, a function call is initiated, and the return value of the function is used as the content of Observation. Action and Observation are appended to the end of Prompt A and Prompt B by the tool manager. At this point, a Thought-Action-Observation loop is finished. This process is repeated continuously until the complex LLM determines that the task is completed during writing Thought. Then it turns to generating the Final Answer. Similarly, the output parser sends the Final Answer to both the user and the memory manager.

3.3. Tool manager

The tool manager is the actual place where the agent interacts with the external environment. It provides a series of predefined functions for the agent to obtain the readings of the sensors and control specific devices through IoT. The tool manager we designed consists of a function library, a function description generator and a function executor, as shown in Figure 3.

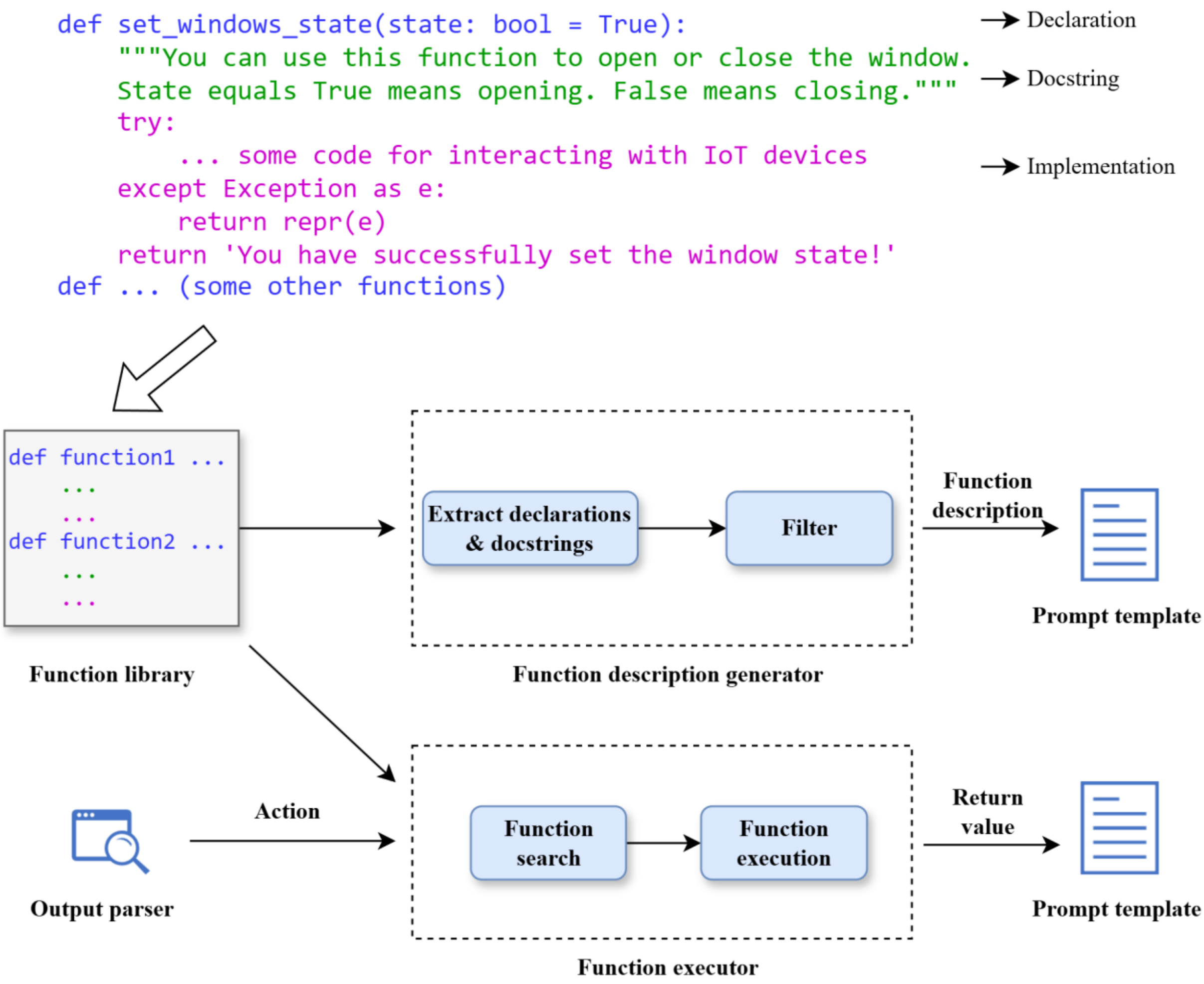

Figure 3. Tool manager.

All functions available for the agent are stored in the function library. A typical example of a Python function is shown in Figure 3. The blue part is the declaration, specifying the function name, input parameters and their types. The green part is the docstring, a unique syntax feature of Python, which follows the declaration and provides further explanation of the function's usage. The purple part is the specific implementation of the function. These functions are highly customizable, allowing for adjustments in the length of their return values based on actual needs. For example, if the sensors in the IoT system are conducting high-frequency sampling, they will output thousands of records at once. On this occasion, when designing functions, developers can adapt to the LLM's context length by filtering, resampling or extracting summary statistics (e.g., mean, max, min) from the records.

In fact, in the tool use process, the LLM do not implement the actions for each task directly. Rather, it only calls and executes the function based on its corresponding declaration and docstring. Since Python is an interpreted and dynamic programming language, we can directly obtain the source code of each function after the program starts with the help of Python's built-in inspect module and extract the declaration and docstring from it. This operation is completed by the function description generator before each Thought-Action-Observation loop. Additionally, some researchers have observed a typical failure mode in ReAct agents, where the LLM repeatedly outputs the same Action and leads to a deadlock [4]. To avoid this problem, we added a filter to the function description generator. If the agent successfully called a function in the previous loop but still repeats the same function in the current loop, it will be filtered out so that it will not appear in the prompt of the next loop, thus helping the agent break out of the deadlock.

The function executor in the tool manager is responsible for receiving Action sent by the output parser, finding the corresponding function object in the function library based on the content of Action, passing in parameters and then executing it. Since the Action is written in JSON format, this operation is relatively easy to implement. After the function execution is completed, the function's return value (e.g., "You have successfully set the window state!" in Figure 3) is put behind the string "Observation:" and the function executor adds the content of Action and Observation to Prompt A and Prompt B.

3.4. Memory manager

Memory manager is designed to record the conversations between the user and the agent, thereby endowing the agent with memory capabilities. A common practice to achieve this is adding the user's question and the agent's Final Answer to a list every time a Final Answer is generated. During the next conversation, the content of the list is filled into the {chat_history} placeholder in the prompt template. However, as the conversation progresses, the prompt becomes increasingly longer, eventually affecting the computational efficiency of the LLM. Although a threshold can be set to automatically delete the earliest message in the message list when the number of messages exceeds the threshold in order to maintain a constant number of elements in the message list, this approach also leads to information loss.

To solve the problem above, we adopt a memory manager that combines short-term memory and long-term memory, as shown in Figure 4. A list for storing messages is created in the program as short-term memory. Meanwhile, a file is created on the hard disk as long-term memory. The short-term memory only contains the chat history of the user and the agent, while the long-term memory includes both the chat history and the dates of conversations. Messages are first saved in short-term memory. When the number of messages exceeds the predefined threshold, the earliest message is moved to long-term memory and saved in the file.

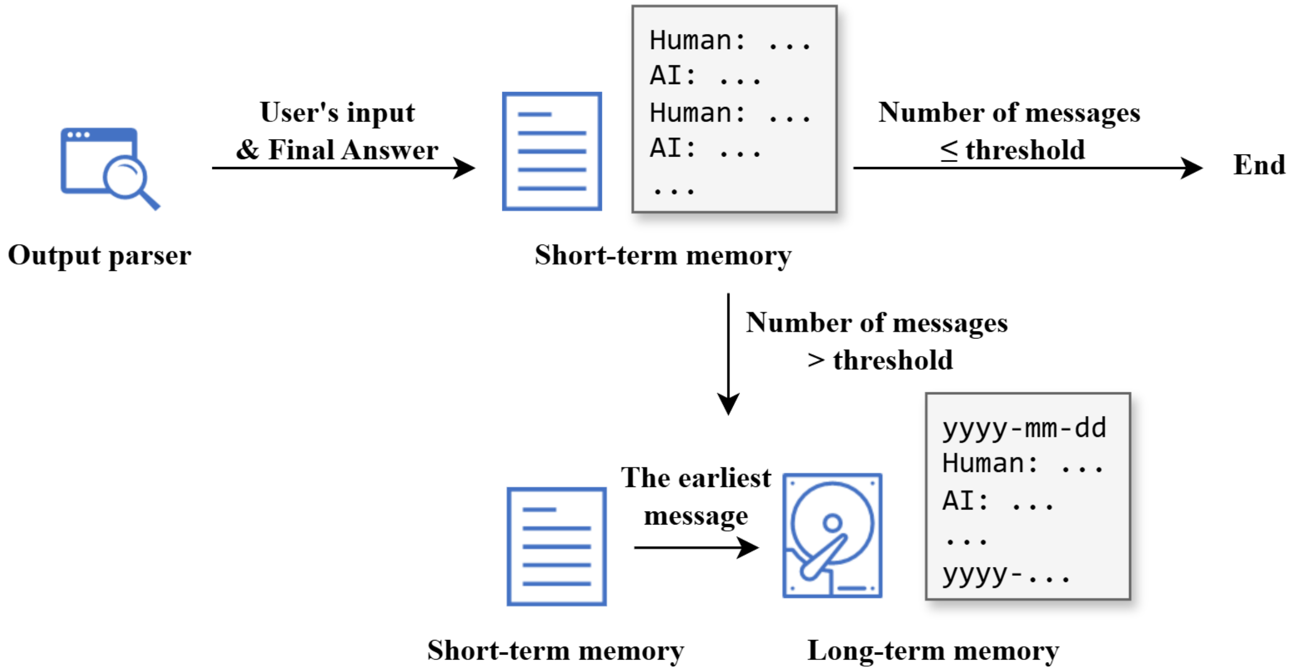

Figure 4. Memory manager.

The content in short-term memory can be directly filled into the prompt template. For accessing long-term memory, tool use mechanism is employed, *i.e.*, adding a "recall" function to the function library, which has three input parameters: the year, month, and day of the conversation. It is used to read the file storing long-term memory and find the chat history for a specific day based on the input date. The content queried is fed into the agent system in the form of a return value for further analysis by the LLM.

4. Experiment

Two experiments were conducted to verify the feasibility and effectiveness of the proposed framework. First, in order to investigate whether the agent can appropriately respond to a wide range of user requests, we constructed a virtual building using Python code, where the agent can interact with various furniture and appliances. The user sent instructions to the agent, and its performance was evaluated by observing its behavior. After that, to verify the compatibility of the agent with the IoT devices in the real world, we built a simulation environment using embedded devices and a model building, where the entire AI system was tested.

4.1. Virtual experiment

We supposed that the virtual building contained an automatic door, an automatic window, a smart air conditioner and a smart light. Besides, two temperature and humidity sensors were mounted on the wall.

One is indoors and the other is outdoors. A series of global variables were created to store the state information of the devices, including:

- Indoor and outdoor temperature and humidity;
- Whether the door and window are open;
- The mode (closed/cooling/dehumidifying), temperature, and wind speed of the air conditioner;
- Whether the air conditioner is locked;
- The mode (closed/quiet/enthusiastic) and brightness of the light.
- Additionally, we wrote some functions to get or set the values of these variables. Figure 5 shows this virtual building in a 3D model and lists the functions callable by the agent.

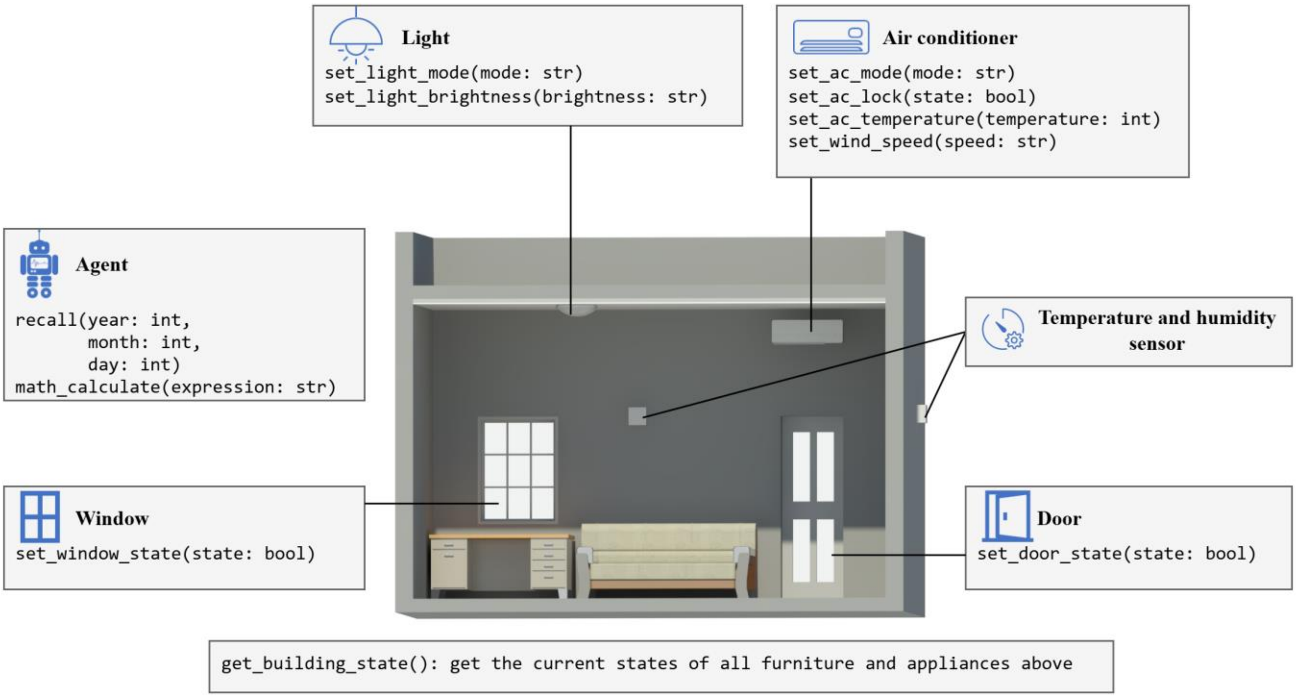

Figure 5. Visual building.

It is worth noting that the set_ac_lock function was designed to test whether the agent could handle abnormal situations. When calling the set_ac_temperature function, if the air conditioner is locked, an error message will be returned, and the agent should call the set_ac_lock function to unlock the air conditioner before proceeding. However, this rule is not included in the function description, so it is not priori knowledge for the agent. The agent can only realize this fact after seeing the error message, so it can test the adaptability of the agent.

We constructed 100 instructions, sent them to the agent in the virtual building, and obtained its thought processes and final answers. Since this study focuses on open-ended human-computer interaction tasks without predefined categories, we evaluate the agent's performance at the final effect level, which is a common practice in generative AI applications [19,20,22,25,26]. For each instruction, the agent's response was manually evaluated. If the agent's operational process did not interrupt and both the process and result were reasonable and logical, the agent's response to the instruction was considered correct.

The test instructions covered six categories: daily conversation, intent recognition, reasoning task, multi-task test, memory test and refusal test. The meanings of each category are as follows:

- Daily conversation: General dialogues between the user and the AI, which do not require any tools and can be completely solved with the inherent knowledge of the LLM alone.
- Intent recognition: The user does not specify which devices they want to control and/or the settings of the devices, only stating a general need. The agent has to understand the meaning and intent of the user's words, extract necessary information, and then call the appropriate tools.
- Reasoning task: The user's instruction is relatively clear, but the agent cannot directly obtain the answer from the tool's return value and should alternate between tool calling and the thought process to complete the task.
- Multi-task test: The user's instruction is relatively clear, but two or more tasks are included in one instruction. The agent's operation is considered successful only when it completes all tasks.
- Memory test: Assigning some tasks that involve chat history. The agent should be able to obtain information from the context of the current chat and call the recall function to query long-term memory if necessary to complete the task.
- Refusal test: Instructions that exceed the agent's capabilities. The agent has to recognize the boundaries of its abilities and refuses the user's request in time, rather than fabricating answers arbitrarily based on its hallucinations. Under the circumstances, the complex LLM is expected to directly reject the request and generate a response such as "I'm sorry, but I don't have access to..." without invoking the lightweight LLM.

Figure 6 shows the proportions of each category and provides some examples. Distinguished from other categories, the memory test consists of 10 pairs and each pair consists of two correlated instructions.

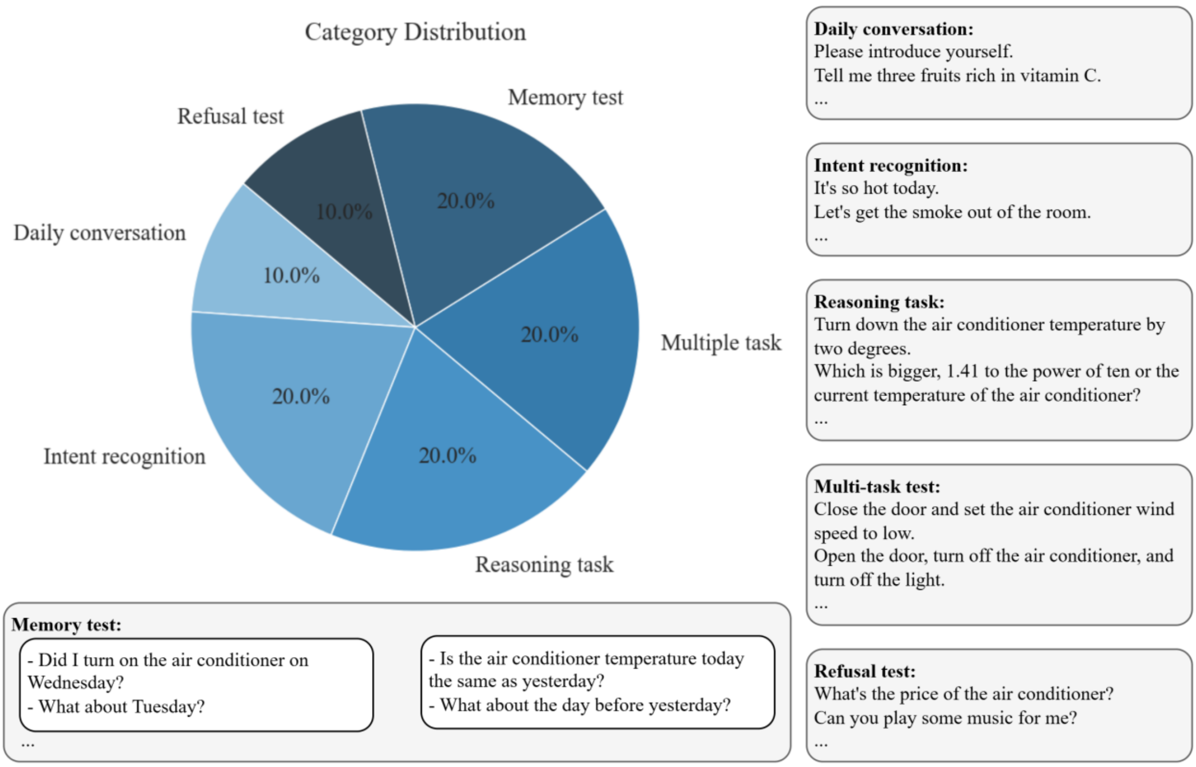

Figure 6. Test instructions.

In the test, we built the agent based on the API provided by ZhipuAI, which is compatible with the OpenAI Python package and can work normally. The complex LLM was GLM-4-Plus, a GPT-4 level LLM, while the lightweight LLM was GLM-4-FlashX, whose price was only 1/500 of the former's. The stop tokens for GLM-4-Plus and GLM-4-FlashX were set to "Action" and "Observation" respectively. The temperature parameters of the two models were set to 0, and the top_p parameters were not specified manually. The date for the conversation was set to October 31, 2024. The number of the agent's maximum iteration steps was set to 10, *i.e.*, if the agent still cannot complete the task after 10 loops, its operation was considered failed. The test data were input into the agent for inference, and its accuracy was statistically analyzed. Additionally, we tested a scheme only using the complex LLM, where the "format constraints" section in Table 2 was merged into the prompt template of complex LLM. The final test results are shown in Figure 7.

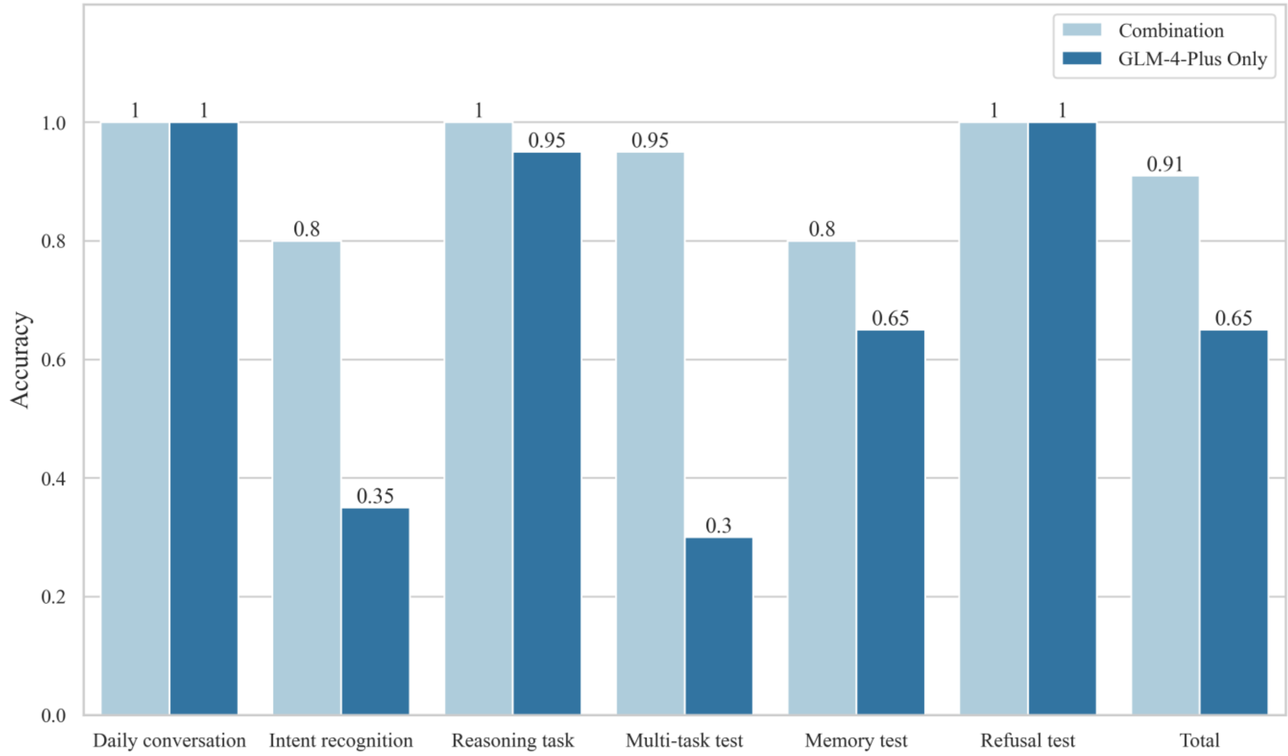

Figure 7. Test accuracy.

It can be seen that the dual-model collaboration scheme exhibits outstanding performance, achieving 100% accuracy in daily conversation, reasoning task and refusal test and the accuracy in multi-task test can also reach 95%. The accuracy of the other two tasks decreases somewhat, but it still does not fall below 80%.

Figure 8 presents two typical examples, demonstrating the LLM's capabilities of intent recognition, reasoning and anomaly handling.

In the left part of Figure 8, the user wanted to remove the smoke in the room. To accomplish this task, the agent underwent five rounds of loop, as follows:

- Round 1: The agent understood the user's intent and called the get_building_state function to check the status of the doors, windows, and air conditioner.

- Round 2: The agent observed the return value from Round 1 and called the set_window_state function to open the window.
- Round 3: The agent called the set_door_state function to open the door.
- Round 4: The agent called the set_wind_speed function to adjust the air conditioner wind speed to high.
- Round 5: The agent recognized that the window, door and air conditioner had all been successfully adjusted to appropriate states and generated the final answer.

In the right part of Figure 8, the user commanded the agent to turn down the air conditioner temperature by two degrees. The agent also went through five rounds of loop:

- Round 1: The agent noticed that the prerequisite for the task was to ascertain the current air conditioner temperature, so it called the get_building_state function to retrieve this information.
- Round 2: The agent learned that the current air conditioner temperature was 29 degrees, which minus 2 degrees resulted in 27 degrees. It passed 27 as an input parameter to the set_ac_temperature function. However, since the air conditioner lock was not open, an error message occurred.
- Round 3: The agent called the set_ac_lock function to unlock the air conditioner.
- Round 4: The agent attempted to call the set_ac_temperature function again and successfully adjusted the temperature to 27 degrees.
- Round 5: The agent recognized that the task was completed and generated the final answer.

Figure 9 presents two response examples related to memory, showing the agent's ability to handle tasks which require access to historical data. In this example, the user engaged in two rounds of dialogue with the agent. The agent successfully recalled past conversations according to "yesterday" and "the day before yesterday", and integrated the information to provide coherent and contextually appropriate responses.

However, although the agent using the complex LLM alone still performed well in daily conversation, reasoning task, and refusal test, its performance was seriously deteriorated in the other three tasks. The accuracy in multi-task test even fell to only 30%. The overall accuracy was only 65%, significantly lower than the 91% of the combination scheme. By observing the agent's outputs, it was found that the main reason for this result was that the LLM did not follow the rule "Only one JSON should be included in your response" in the prompt and output multiple Actions continuously after the Thought (as shown in Figure 10), which caused program error. This issue frequently occurred in many tests, ultimately leading to a substantial decline in accuracy. Although a more complex output parser could be designed to extract only the first JSON and discard the rest content, thereby solving this problem, this approach would obviously result in token waste.

Human message

Let's get the smoke out of the room.

AI message

Thought: The user wants to remove smoke from the room. To achieve this, I need to consider actions like opening windows and doors, and possibly adjusting the air conditioner. I'll start by checking the current state of the building to determine the best course of action.

Action:
```json
{
  "action": "get_building_state",
  "action_inputs": []
}
```

Observation:
Indoor temperature: 30 degrees
...
Window: closed
Door: closed
...
Air conditioner wind speed: medium

Thought: ... I should open both the windows and the door to allow fresh air to circulate. Additionally, ... I might need to adjust the wind speed to high to enhance the air flow.

Action:
```json
{
  "action": "set_window_state",
  "action_inputs": [true]
}
```

Observation: You have successfully set the window state!

Action:
```json
{
  "action": "set_door_state",
  "action_inputs": [true]
}
```

Observation: You have successfully set the door state!

Thought: The windows and doors are now open ... I should set the air conditioner wind speed to high.

Action:
```json
{
  "action": "set_wind_speed",
  "action_inputs": ["high"]
}
```

Observation: You have successfully set the air conditioner wind speed!

Thought: With the windows and doors open and the air conditioner wind speed set to high, the room should start to ventilate effectively, helping to remove the smoke. I now know the final answer.

Final Answer: The room is now being ventilated to remove the smoke. The windows and doors have been opened, and the air conditioner wind speed has been set to high to enhance air circulation.

Human message

Turn down the air conditioner temperature by two degrees.

AI message

Thought: The user wants to decrease the air conditioner temperature by two degrees. I need to first check the current temperature setting of the air conditioner.

Action:
```json
{
  "action": "get_building_state",
  "action_inputs": []
}
```

Observation:
Indoor temperature: 30 degrees
...
Air conditioner temperature: 29 degrees
Air conditioner wind speed: medium

Thought: The current air conditioner temperature is 29 degrees. To decrease it by two degrees, I need to set it to 27 degrees.

Action:
```json
{
  "action": "set_ac_temperature",
  "action_inputs": [27]
}
```

Observation: Error! The air conditioner is locked. Please unlock it first.

Thought: The air conditioner is currently locked ... I need to unlock it before I can set the new temperature.

Action:
```json
{
  "action": "set_ac_lock",
  "action_inputs": [false]
}
```

Observation: You have successfully set the air conditioner lock state!

Thought: Now that the air conditioner is unlocked, I can proceed to set the temperature to 27 degrees as requested by the user.

Action:
```json
{
  "action": "set_ac_temperature",
  "action_inputs": [27]
}
```

Observation: You have successfully set the air conditioner temperature!

Thought: I have successfully decreased the air conditioner temperature by two degrees as requested by the user.
Final Answer: The air conditioner temperature has been set to 27 degrees.

Figure 8. Two response examples not related to memory.

Human message

Is the air conditioner temperature today the same as yesterday?

AI message

Thought: To determine if the air conditioner temperature today is the same as yesterday, I need to retrieve the current temperature setting and compare it with the setting from yesterday. I will first get the current state of the building to find today's air conditioner temperature.

Action:

```json
{
  "action": "get_building_state",
  "action_inputs": []
}
```

Observation:

...

Thought: I have the current air conditioner temperature, which is 29 degrees. Now, I need to retrieve the air conditioner temperature setting from yesterday to compare.

Action:

```json
{
  "action": "recall",
  "action_inputs": [2024, 10, 30]
}
```

Observation:

2024-10-30

Human: Turn up the air conditioner temperature by one degrees.

AI: The air conditioner temperature has been set to 27 degrees.

Human: Set the air conditioner wind speed to high.

AI: ...

...

Thought: From the chat history of yesterday, I can see that the air conditioner temperature was set to 27 degrees. Today's air conditioner temperature is 29 degrees. Therefore, the temperature is not the same as yesterday.

Final Answer: No, the air conditioner temperature today is not the same as yesterday. Today it is set to 29 degrees, while yesterday it was set to 27 degrees.

Human message

What about the day before yesterday?

AI message

Thought: To answer the user's question about the air conditioner temperature the day before yesterday, I need to retrieve the chat history for that date. This will allow me to check the recorded temperature setting for the air conditioner.

Action:

```json
{
  "action": "recall",
  "action_inputs": [2024, 10, 29]
}
```

Observation:

2024-10-29

Human: I want to make some sandwiches. What ingredients do I need to buy?

AI: To make sandwiches, you typically need ...

...

Human: Turn down the air conditioner temperature by two degrees.

AI: The air conditioner temperature has been successfully set to 24 degrees.

...

Thought: Based on the chat history for the day before yesterday (October 29, 2024), the air conditioner temperature was set to 24 degrees. Now I can compare this with the temperatures from yesterday and today to provide a complete answer to the user.

Final Answer: The day before yesterday, the air conditioner temperature was set to 24 degrees. So, compared to yesterday's 27 degrees and today's 29 degrees, it was cooler the day before yesterday.

Figure 9. Two response examples related to memory.

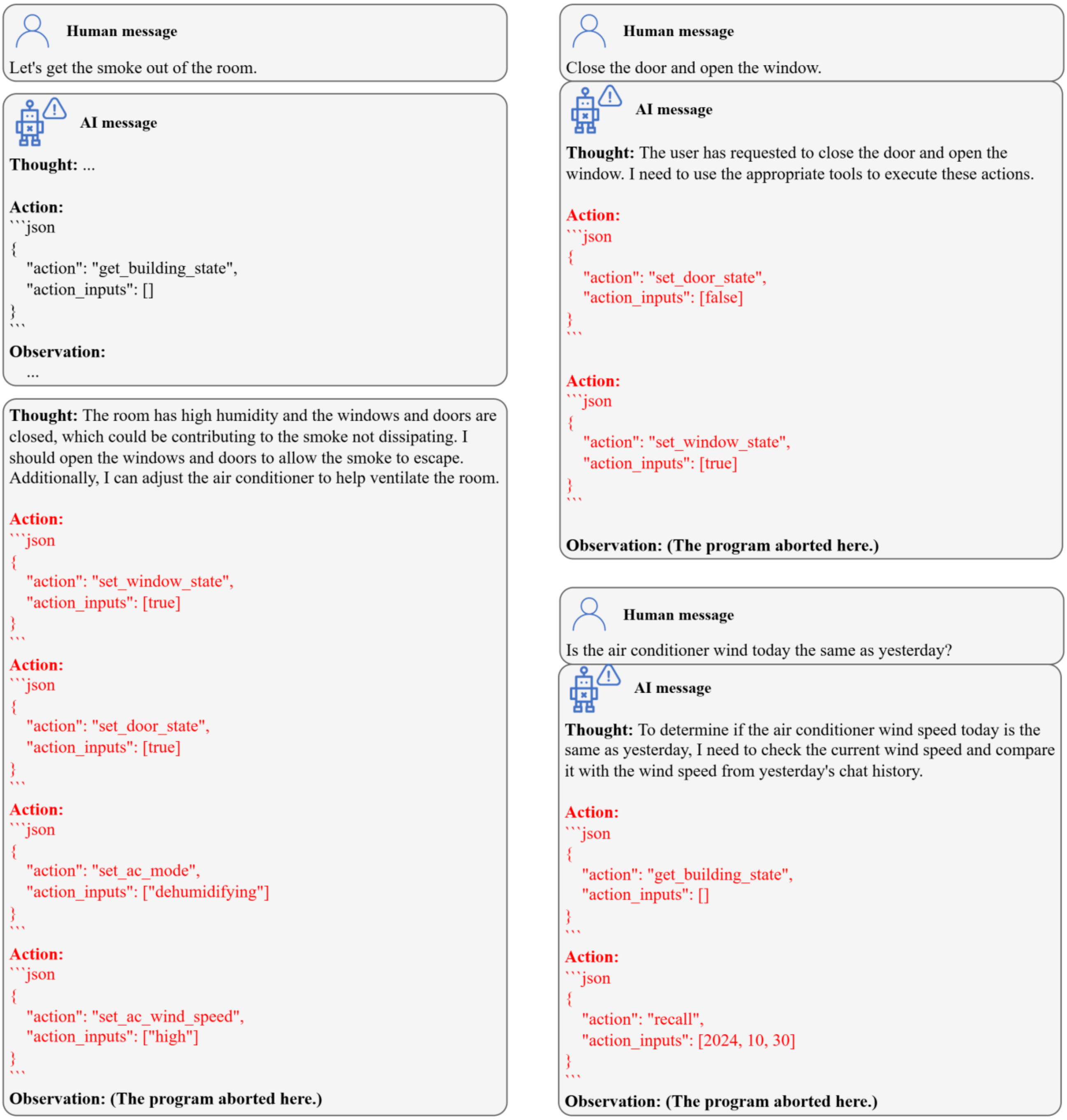

Figure 10. The main failure mode of agent with GLM-4-Plus only.

In order to test the agent's ability in handling self-contradictory instructions, we provided it with the request "Turn down the air conditioner temperature by ten degrees and make the room warmer". As shown in Figure 11, the agent successfully recognized the contradiction and informed the user without generating any device control actions.

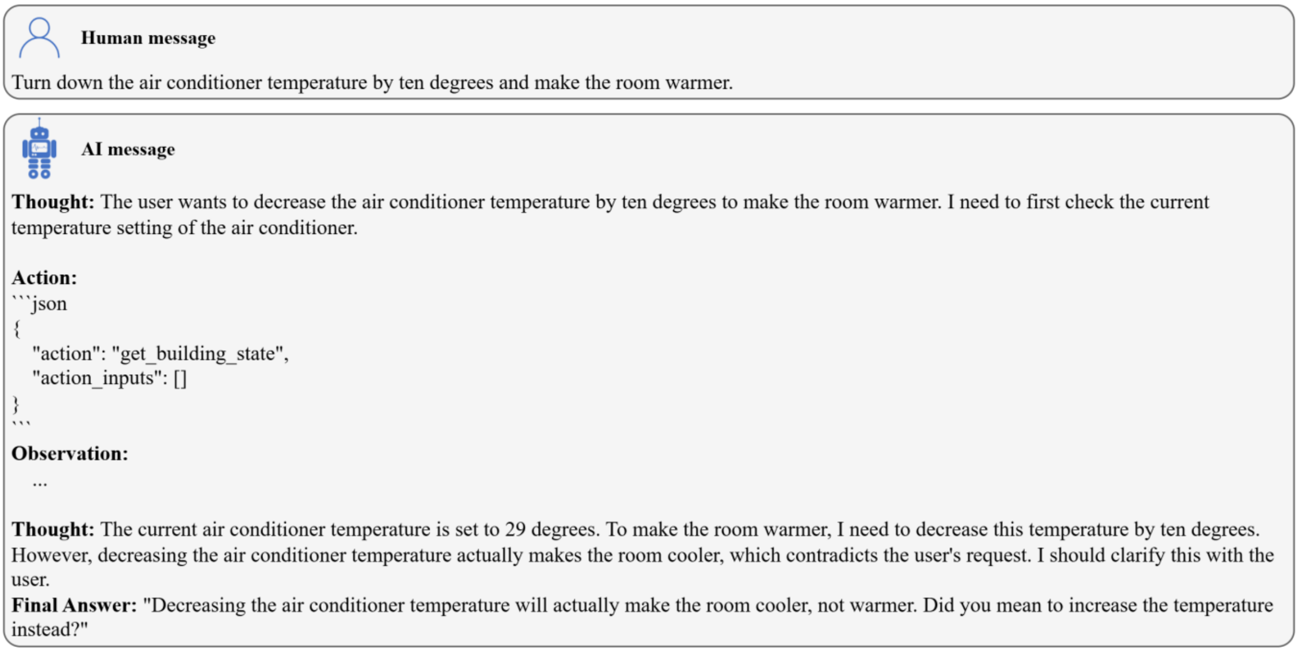

Figure 11. A response example of handling self-contradictory instructions.

4.2. Model experiment

We installed some embedded devices in a wooden building model to form a simulated smart building. The proposed agent was implanted into the system as a control module to verify whether it can collaborate with the existing IoT devices. Due to hardware limitations, the configuration of the model building is not identical to that of the virtual building. In the model building, the air conditioner was replaced with an electric fan, and the number of temperature and humidity sensors was reduced from two to one, which was placed in the room. Therefore, the tools were also simplified.

The composition of the smart building system is shown in Figure 12. The ESP32 MCUs act as the lower computers, which are directly connected to the door, window, fan, light, and temperature and humidity sensor. The Raspberry Pi 5 serves as the upper computer, running three modules: web server, voice interaction module and AI agent. The web server can send instructions to the ESP32 MCUs via WiFi. The voice interaction module makes the simulation environment more similar to real-world usage scenarios. To achieve this functionality, we connected a USB microphone and a USB loudspeaker to the Raspberry Pi 5 and deployed keyword spotting, speech recognition and speech synthesis program on it. The keyword spotting and speech recognition program run locally, using the fsmn-ctc model [33] and Paraformer model [34] respectively, while the speech synthesis program uses Huawei Cloud’s online service. The AI agent adopts the GLM-4-Plus and GLM-4-FlashX combination scheme described in Section 4-1, depending on the cloud service provided by ZhipuAI.

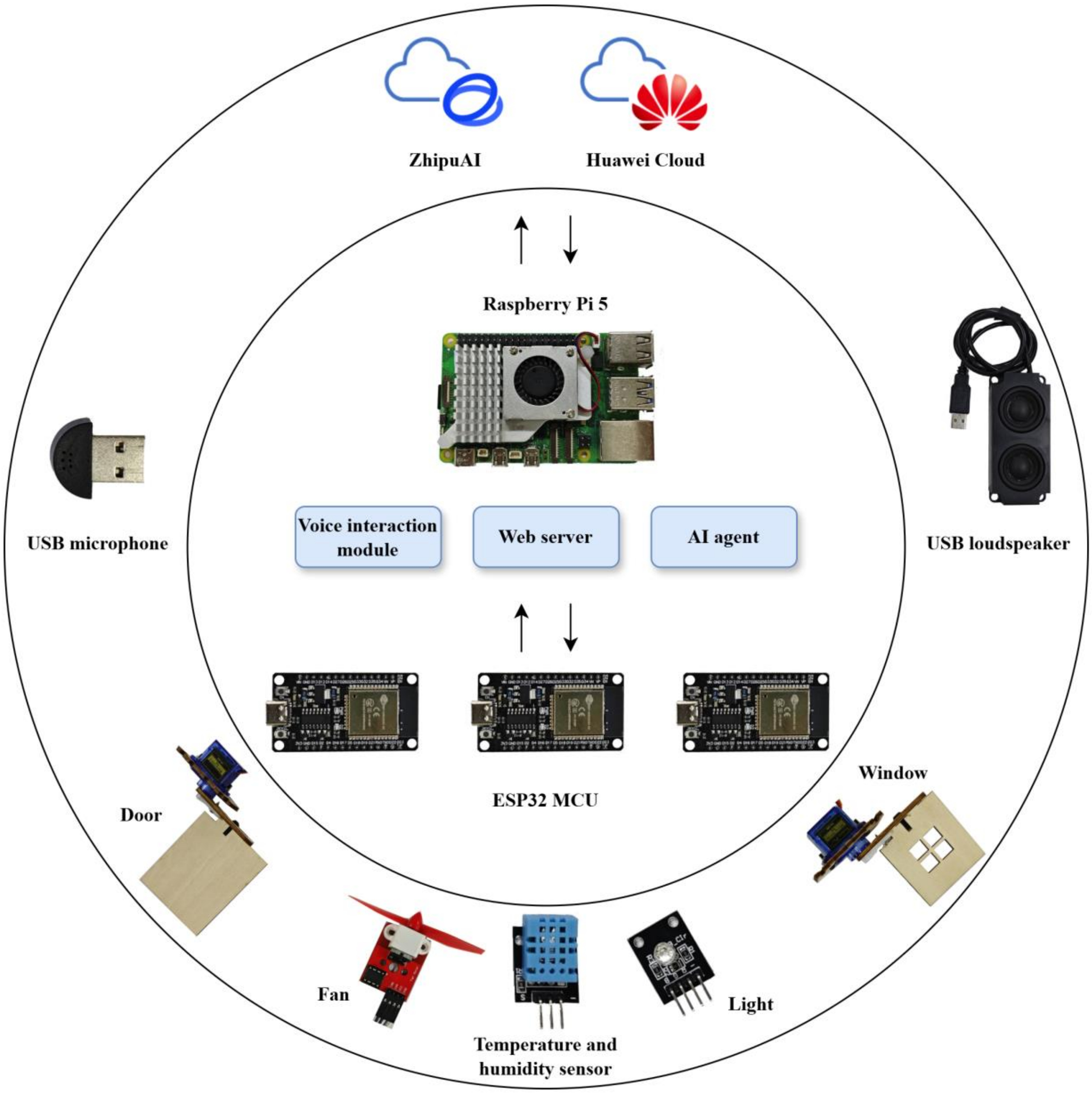

Figure 12. The composition of the smart building system.

The test results showed that the AI agent exhibited excellent compatibility with IoT devices and could function normally in the smart building. Figure 13 presents a typical example of the system's operation. By default, the system was in sleep mode, with only the keyword spotting program in the voice interaction module running to minimize power consumption. This program continuously collected sound information from the environment and analyzed whether the wake-up word was heard. When the user said "Hi, Xiaowen," the system was awakened and replied to the user in voice: "I'm here!" Then, it entered the working stage. At this point, the user said: "The room feels a bit stuffy." This audio was input into the speech recognition program, converted into text, and provided to the AI agent. Then, the agent entered the Thought-Action-Observation loop. First, it decided to call the get_building_state function to obtain the current state data of the building. After reviewing the data, to improve indoor ventilation, the agent decided to call the set_window_state function to open the window, then it called the set_fan_state function to turn on the fan, and finally it generated a final answer to notify the user that the operation was completed. The content of the final answer was input into the speech synthesis program and converted to audio, which was then played through the loudspeaker.

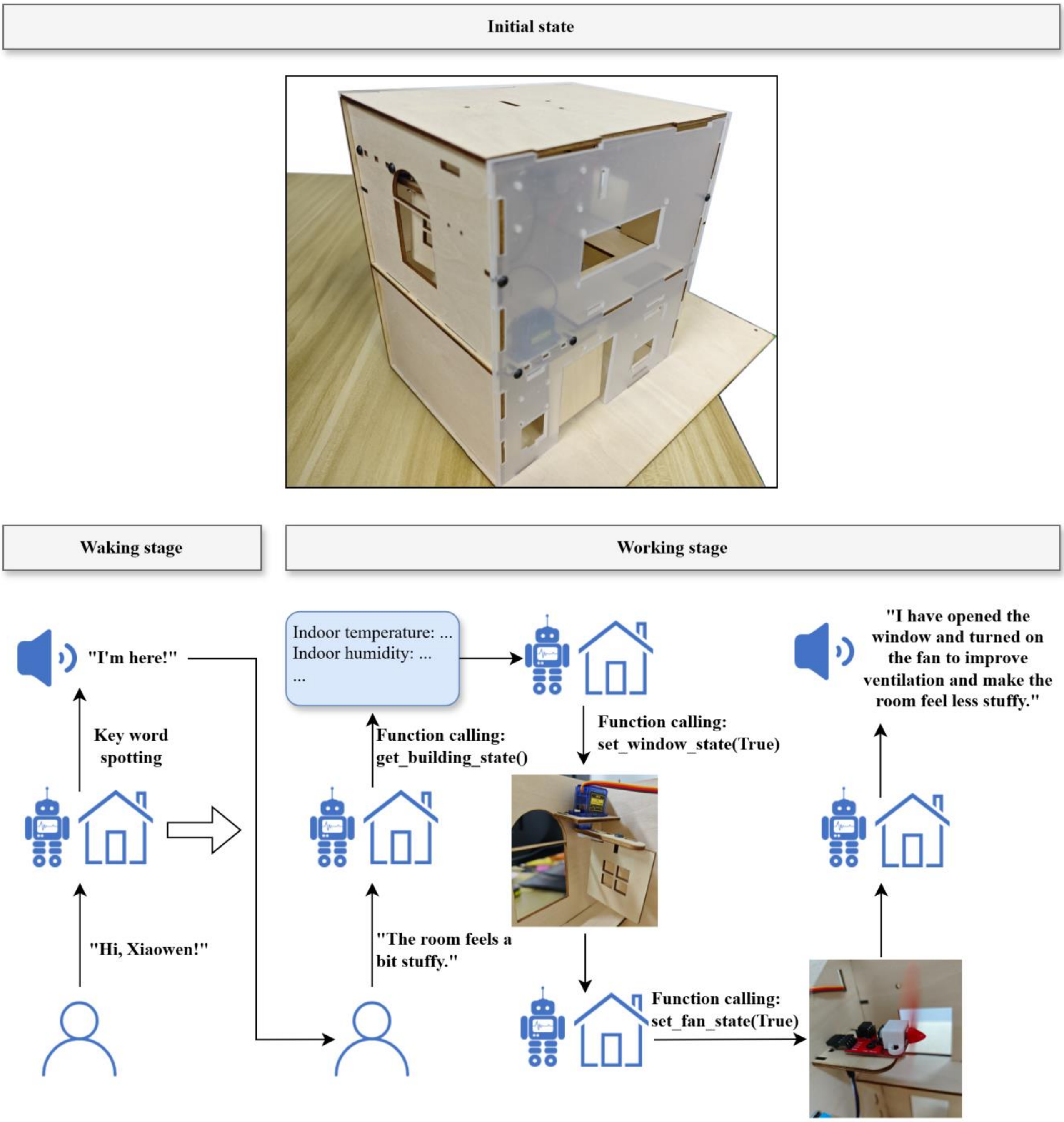

Figure 13. An example of the system running process.

5. Conclusions and future work

To improve the accuracy and convenience of human-computer interaction in smart buildings, this paper introduces large language models and AI agents into smart building systems, designs various components and forms an AI agent based on the ReAct strategy. Two experiments have verified its feasibility and effectiveness. Our main contributions are as follows:

- We proposed a modular and general AI agent framework for building management, which enables the smart building systems to perform complex, humanized and tool-based interactions.
- The experiment conducted in a virtual building showed that the proposed agent can not only engage in daily conversations like an ordinary chatbot but also exhibit outstanding capabilities in intent recognition, reasoning, using multiple tools, contextual understanding, recalling and refusing, which achieved an accuracy of 91% in a simulated human-computer interaction test.
- The proposed agent was deployed in a real-world model building. The experimental results demonstrated the tool manager provided a universal interface between the agent and various IoT devices, enabling the agent to be integrated into existing IoT systems and creating conditions for the practical application of this technology.

However, this study still has some limitations. We only discussed some simple usage scenarios in smart buildings, and further research is needed on the application of LLMs and agents in more complex scenarios, such as fire evacuation and building security. To handle these complex tasks, fine-tuning the LLM to introduce more domain knowledge and reasoning capabilities may be necessary. For the proposed agent prototype, only one action is allowed in a round, but parallel function calling is a way to improve the agent's operation efficiency. We plan to explore mechanisms which enable the LLM to identify whether causal relationships are not a concern for multiple actions and initiate parallel function calling if possible. Although the proposed agent is theoretically applicable to any building with IoT systems, conducting further tests across different types and scales of buildings is still necessary. Other considerations include integrating LLMs with edge computing to reduce dependency on Internet accessibility, and exploring the application of multimodal large language models to enhance the agent's information acquisition capability.

Acknowledgments

This work was supported by Key Technologies R&D Program (Grant No. 2022YFC3801203), supported by National Natural Science Foundation of China (Grant No. 52108286), supported by Shenzhen Science and Technology Programs (Grant No. GXWD20220818002513001, Grant No. RCBS20221008093128076, Grant No. ZDSYS20210929115800001) and supported by Key Laboratory Open Fund Project of Shenzhen Technology Institute of Urban Public Safety, supported by Young Elite Scientists Sponsorship Program by CAST (2023QNRC001).

Conflicts of interests

The authors declare that they have no conflicts of interest like employment, consulting fees, research contracts, stock ownership, patent licenses, honoraria, advisory affiliations *etc.*

Authors' contribution

Xiangjun Yan: Conceptualization, Methodology, Software, Data curation, Writing - original draft. Xincong Yang: Conceptualization, Funding acquisition, Writing - review & editing, Supervision. Nan Jin: Resources, Investigation, Data curation, Validation. Yu Chen: Software, Formal analysis, Investigation, Visualization. Jiaqi Li: Resources, Investigation, Data curation.

References

[1] Habash R. Building as a system. In *Sustainability and Health in Intelligent Buildings*, Kidlington: Woodhead Publishing, 2022, pp. 1–32.

[2] Iliev Y, Ilieva G. A Framework for Smart Home System with Voice Control Using NLP Methods. *Electronics* 2023, 12(1):116.

[3] Jia M, Komeily A, Wang Y, Srinivasan RS. Adopting Internet of Things for the development of smart buildings: A review of enabling technologies and applications. *Autom. Constr.* 2019, 101:111–126.

[4] Yao S, Zhao J, Yu D, Du N, Shafran I, *et al*. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations (ICLR)*, Kigali, Rwanda, 1–5 May, 2023.

[5] Rashid KM, Louis J, Fiawoyife KK. Wireless electric appliance control for smart buildings using indoor location tracking and BIM-based virtual environments. *Autom. Constr*. 2019, 101:48-58.

[6] Vogiatzidakis P, Koutsabasis P. 'Address and command': Two-handed mid-air interactions with multiple home devices. *Int. J. Hum.-Comput. Stud.* 2022, 159:102755.

[7] Park S, Wang X, Chen K, Lee Y. Station: Gesture-Based Authentication for Voice Interfaces. *IEEE Internet Things J.* 2024, 11(12):22668–22683.

[8] Wang Y, Han Y, Wu Y, Korkina E, Zhou Z, *et al.* An occupant-centric adaptive façade based on real-time and contactless glare and thermal discomfort estimation using deep learning algorithm. *Build. Environ.* 2022, 214:108907.

[9] Klaib AF, Alsrehin NO, Melhem WY, Bashtawi HO. IoT Smart Home Using Eye Tracking and Voice Interfaces for Elderly and Special Needs People. *J. Commun*. 2019, 14:614–621.

[10] Zhang R, He S, Yang X, Wang X, Li K, *et al.* An EOG-Based Human–Machine Interface to Control a Smart Home Environment for Patients With Severe Spinal Cord Injuries. *IEEE Trans. Biomed. Eng.* 2019, 66(1):89–100.

[11] Yang C. Design of Smart Home Control System Based on Wireless Voice Sensor. *J. Sens.* 2021, 2021(1):8254478.

[12] Tiwari V, Hashmi MF, Keskar A, Shivaprakash NC. Virtual home assistant for voice based controlling and scheduling with short speech speaker identification. *Multimed. Tools Appl*. 2020, 79(7):5243–5268.

[13] Peña-Cáceres O, Silva-Marchan H, Albert M, Gil M. Recognition of Human Actions through Speech or Voice Using Machine Learning Techniques. *Comput. Mat. Contin.* 2023, 77(2):1873–1891.

[14] Stefanenko OS, Lipinskiy LV, Polyakova AS, Khudonogova JA, Semenkin ES. An intelligent voice recognition system based on fuzzy logic and the bag-of-words technique. *IOP Conference Series: Materials Science and Engineering* 2022, 1230(1):012020.

[15] Shin S, Lee C, Issa Raja RA. Framework for Automatic Speech Recognition-Based Building Information Retrieval from BIM Software. *Constr. Res. Congr. 2020* 2020, 992–1000

[16] Desot T, Portet F, Vacher M. Towards End-to-End spoken intent recognition in smart home. In *2019 International Conference on Speech Technology and Human-Computer Dialogue (SpeD)*, Timisoara, Romania, October 10–12, 2019, pp. 1–8.

[17] Linares-Garcia DA, Roofigari-Esfahan N, Pratt K, Jeon M. Voice-Based Intelligent Virtual Agents (VIVA) to Support Construction Worker Productivity. Autom. *Constr.* 2022, 143:104554.
[18] Brown T, Mann B, Ryder N, Subbiah M, Kaplan JD, *et al*. Language Models are Few-Shot Learners. *Adv. Neural Inf. Process. Syst.* 2020, 33:1877–1901.
[19] Xu Z, Wang Z, Ren P, Zhang X, Li T. Framework and construction methodology of underground engineering domain knowledge large language model: UndergrGPT. *Smart Constr.* 2024, 1(2):0012.
[20] Pu H, Yang X, Li J, Guo R. AutoRepo: A general framework for multimodal LLM-based automated construction reporting. *Expert Syst. Appl.* 2024, 255:124601.
[21] Amer F, Jung Y, Golparvar-Fard M. Transformer machine learning language model for auto-alignment of long-term and short-term plans in construction. Autom. *Constr.* 2021, 132:103929.
[22] Hussain R, Sabir A, Lee D-Y, Zaidi SFA, Pedro A, *et al.* Conversational AI-based VR system to improve construction safety training of migrant workers. Autom. *Constr.* 2024, 160:105315.
[23] Zhang C, Zhang J, Zhao Y, Lu J. Automated data mining framework for building energy conservation aided by generative pre-trained transformers (GPT). *Energy Build.* 2024, 305:113877.
[24] Ahmadi E, Muley S, Wang C. Automatic construction accident report analysis using large language models (LLMs). *J. Intell. Constr.* 2024.
[25] Ghasemi A, Dai F. Can ChatGPT assist in cost analysis and bid pricing in construction estimating? A pilot study using a bridge rehabilitation project. *Smart Constr.* 2024, 1(2):0009.
[26] Wong S, Zheng C, Su X, Tang Y. Construction contract risk identification based on knowledge-augmented language models. *Comput. Ind.* 2024, 157–158:104082.
[27] Wang L, Ma C, Feng X, Zhang Z, Yang H, *et al.* A survey on large language model based autonomous agents. *Front.. Comput. Sci.* 2024, 18(6):186345.
[28] Nakano R, Hilton J, Balaji S, Wu J, Ouyang L, *et al.* WebGPT: Browser-assisted question-answering with human feedback. *arXiv 2022*, arXiv:2112.09332.
[29] Wei J, Wang X, Schuurmans D, Bosma M, ichter b, *et al.* Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Adv. Neural Inf. Process. Syst.* 2022, 35:24824–24837.
[30] Vemprala SH, Bonatti R, Bucker A, Kapoor A. ChatGPT for Robotics: Design Principles and Model Abilities. *IEEE Access 2024*, 12:55682-55696.
[31] Chase H. react-chat. 2023. Available: https://smith.langchain.com/hub/hwchase17/react-chat (accessed on 30 November 2024).
[32] Chase H. react-json. 2023. Available: https://smith.langchain.com/hub/hwchase17/react-json (accessed on 30 November 2024).
[33] Chen M, Zhang S, Lei M, Liu Y, Yao H, *et al.* Compact Feedforward Sequential Memory Networks for Small-footprint Keyword Spotting. In *Interspeech 2018*, Hyderabad, India, September 2–6, pp. 2663–2667.
[34] Gao Z, Zhang S, McLoughlin I, Yan Z. Paraformer: Fast and Accurate Parallel Transformer for Non-autoregressive End-to-End Speech Recognition. In *Interspeech 2022*, Incheon, Korea, September 18–22, pp. 2063–2067.