BIM Design Assistant Driven by LLM Agents

Jin Han[1], Biansining Zhou[2], Xin-Zheng Lu[1], Zhen-Zhong Hu[3], Jun Ma[4] and Jia-Rui Lin[1, *]

[1]Department of Civil Engineering, Tsinghua University, China
[2]Department of Computer Science and Technology, Tsinghua University, China
[3]Institute for Ocean Engineering, Shenzhen International Graduate School, Tsinghua University, Shenzhen, China
[4]Department of Urban Planning and Design, University of Hong Kong, Hong Kong
Corresponding author: lin611@tsinghua.edu.cn

Abstract –

BIM design workflows involve complex multi-step interactions with professional software, limiting the direct use of large language models (LLMs). This paper proposes an LLM-agent-driven BIM design assistant that enables natural-language-based BIM design and visualization through reliable tool invocation. First, a two-layer Revit interface function library is designed to bridge LLM agents and BIM software, providing secure, extensible, and fine-grained model manipulation capabilities. Based on the ReAct paradigm, an agent framework is developed that integrates dialogue history management, structured prompt engineering, and step-by-step reasoning to support long-horizon and multi-turn BIM tasks. An agent-oriented BIM benchmark is constructed to evaluate the system, covering querying, visualization, modification, and deletion tasks with varying difficulty. Experiments with three representative LLMs show that high-performance models can reliably act as BIM design assistants within the proposed framework, achieving up to 90% task accuracy. The results demonstrate the feasibility of LLM-driven BIM design assistance.

Keywords –

Building Information Modeling; Revit; Large language models; Agent

1 Introduction

Building Information Modeling (BIM) has become a key data-driven approach across the building lifecycle [1]. By integrating project data into a unified model, it improves collaboration and information management efficiency [2]. However, as BIM models become more complex, tasks such as model manipulation, information retrieval, and modification still rely heavily on manual interaction with tools like Autodesk Revit [3], increasing barriers for non-expert users and reducing efficiency in repetitive multi-step operations.

More recently, advances in natural language processing and large language models (LLMs) have stimulated growing interest in language-driven BIM interaction [4]. Existing work has investigated LLM-based methods for NLP-based information extraction and analysis [5,6,7], knowledge acquisition [8,9], BIM question answering [10], and building information modeling [11]. However, most of these approaches focus on semantic-level tasks where LLMs mainly support information understanding or text generation. Their capability to execute design operations within BIM software environments remains limited.

The emergence of LLM-based agents provides a potential solution by enabling models to reason and interact with external tools [12,13]. In the BIM domain, this development suggests a shift from passive language assistance toward active task execution. For example, Ying et al. proposed a ReAct-based agent for design checking that can interpret requirements and retrieve BIM data [14]. Jang et al. explored natural-language-driven architectural detailing assistance [15]. However, these studies either rely on predefined task rules or support only limited BIM operations, and they do not establish a general framework for reliable natural-language-driven BIM design interaction.

To address this limitation, this study develops a BIM design assistant driven by LLM agents to enhance BIM-assisted design and visualization tasks. The main contributions are threefold: (1) a two-layer BIM interface architecture that converts low-level Revit APIs into structured tools callable by LLM agents; (2) an agent-based interaction framework integrating ReAct reasoning, prompt engineering, and dialogue history management for multi-step BIM operations; and (3) a benchmark containing 100 BIM tasks across querying, visualization, modification, and deletion to systematically evaluate the proposed framework. Although the present implementation is developed in Autodesk Revit, the proposed framework is not entirely Revit-specific. The platform-dependent components are the BIM operation layer, which includes low-level Revit functions and composed Revit Interfaces. By contrast, the two-layer architectural organization, the ReAct-based planning and execution workflow, etc., are platform-agnostic at the

methodological level. Therefore, transferring the framework to other BIM platforms mainly requires replacing the low-level tool layer with interfaces built on the corresponding software APIs.

2 Design and Implementation of an LLM-Oriented Revit Interface Library

To enable real-time BIM model manipulation, a structured Revit interface library is developed using the official Revit API in C#, exposing core modeling operations to LLM agents and supporting safe agent-based BIM interaction.

2.1 Overall Design

Given the broad functional scope of Revit, this study focuses on structural design tasks and targets common operations such as element retrieval, modification, deletion, parameter configuration, and visualization.

To support these operations efficiently and safely, we design a two-layer interface architecture, as illustrated in Figure 1. The lower layer, Revit Functions, encapsulates low-level BIM model operations and directly interfaces with the Revit. While the upper layer, Revit Interfaces, composes basic functions into higher-level, agent-facing operations, enabling more expressive manipulation.

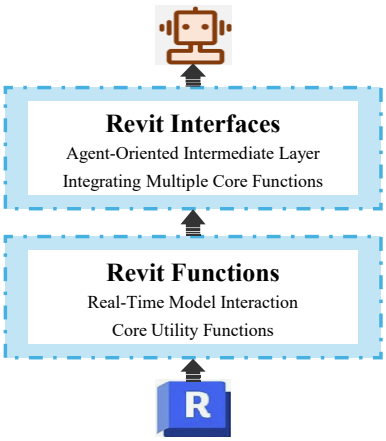

Figure 1. Two-layer interface architecture

This layered design allows shared functionality to be implemented once at the lower level while supporting diverse, integrated operations at the upper level, thereby improving extensibility and development efficiency. Moreover, since all model modifications in Revit are managed through transaction mechanisms, potentially unsafe operations are isolated at the lower layer and exposed to the agent only under controlled conditions at the upper layer. Structured input-output descriptions further accompany each Revit interface to facilitate accurate tool selection and invocation by LLM agents.

2.2 Design of Revit Functions

Revit Functions form the low-level interaction layer that directly and safely operate on the BIM model, comprising nine core functions that provide essential modeling capabilities, as illustrated in Table 1.

Table 1. Core Revit functions

Function Name	Description
Convert Unit	Converts parameter value between user and internal unit
Create Element Filter	Builds filters for element selection
Delete Object	Deletes specified model element
Get Category	Retrieves built-in element categories
Get Parameter Unit	Retrieves internal parameter unit
Retrieve Object	Retrieves elements matching given filters
Retrieve Object Params	Retrieves parameters of a specified element
Set Element Parameter	Sets values of editable element parameter
Show Elements	Highlights selected element

When composed appropriately, these core functions are sufficient to support all target operations while providing controlled access to internal model data. And they are designed to be extensible, enabling future tool integration without altering the core framework.

2.3 Design of Revit Interfaces

While Revit Functions provide low-level capabilities, their API-specific data types are not suitable for direct LLM interaction. Therefore, a higher-level layer, Revit Interfaces, composes multiple functions into integrated operations with simplified inputs (e.g., text and numeric values) for agent invocation (Figure 2). These interfaces also return structured execution feedback for reasoning. Eight interfaces are implemented to support the target BIM operations (Table 2).

Table 2. Core Revit interfaces

Function Name	Description
Delete Element	Deletes elements specified by IDs. Return deleted element IDs or the failure cause
Retrieve Element	Return elements matching given filters or the failure cause
Retrieve All Parameters	Return all parameters of a specified element in a structured format or the failure cause
Retrieve	Return a specific parameter using

Parameter With Built-In Id	its built-in ID in a structured format, or the failure cause
Retrieve Parameter With Parameter Name	Return parameters matching a given name in a structured format or the failure cause
Set Parameter With Built-In Id	Sets parameter values by built-in IDs with validity checks. Return parameter IDs and value changes, or the failure cause
Set Parameter With Parameter Name	Sets parameter values by name with validity checks. Return parameter IDs and value changes, or the failure cause
Show Elements	Highlights specified elements in the 3D view. Return displayed elements' IDs, or the failure cause

By leveraging these, the agent can manipulate BIM models through textual inputs. Moreover, the Revit Interfaces layer is designed to be extensible, allowing additional interfaces to be incorporated.

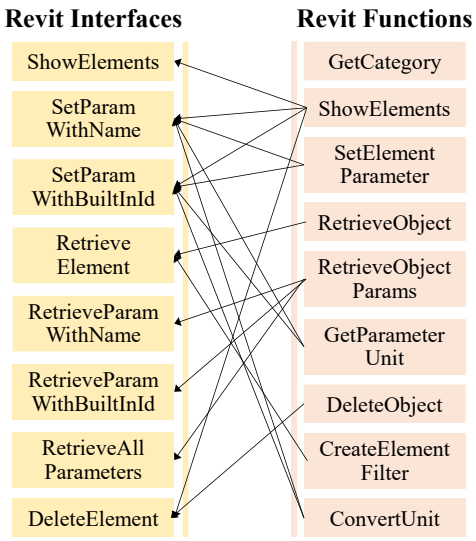

Figure 2. Dependencies between Revit Interfaces and Revit Functions

2.4 Interface Descriptions for LLM-Based Agents

LLM-based tool invocation relies on explicit interface descriptions. Accordingly, each Revit Interface is accompanied by a standardized description consisting of four components: interface name, functionality, input parameters, and output definitions. The interface name serves as the identifier for agent invocation, while the functionality provides a concise summary of the interface behaviour. Input parameters specify the semantics and data types required for correct execution, and output definitions describe the format and meaning of returned information, which supports subsequent agent reasoning and decision-making. These descriptions are assembled during agent-user interactions to support accurate tool selection and execution, as illustrated in Figure 3.

{
"**Interface name**": DeleteElement
"**Functionality**": This interface deletes element(s) from the Revit project given their element ID(s)
"**Input parameters**": The requirements and meanings of the input parameters of this interface are as follows: elementIdStrings: A list of string representations of element IDs of all elements to be deleted
"**Output definitions**": This interface outputs a string message indicating whether the deletion was successful. If failed, a message showing the failure information is returned.
}

Figure 3. Interfaces descriptive information

3 Design Agent Development Based on ReAct and Prompt Engineering

This study adopts the ReAct Think-Act-Observe paradigm as the core framework for BIM-assisted agent development and designs key components based on this paradigm. To ensure the correct interpretation of Revit-specific context, a curated prompt library is incorporated to support reliable multi-turn reasoning and execution.

3.1 Multi-Turn Agent Component Design

Complex BIM design tasks typically require multi-turn interactions with users or the system rather than being completed in a single operation. This necessitates mechanisms for managing dialogue history, controlling the interaction process, and coordinating real-time communication between the agent and Revit interfaces. To meet these requirements, the agent's functionalities are decomposed into modular components that together support a complete conversational workflow.

Specifically, five agent components are designed to collaboratively construct the interaction framework, supporting instruction parsing, task planning, tool invocation, execution, and user feedback, as shown in Figure 4. A dialogue history management system and a planning and execution algorithm form the core of the framework, orchestrating the interaction flow and coordinating other components. Tool invocation is handled by an executor that maps parsed commands to Revit Interfaces, while LLM interaction is supported by a prompt library and LLM APIs for context-aware prompt assembly and response retrieval. The following describes these five components in detail.

1. Dialogue history management system: BIM tasks typically involve long, multi-step interactions,

requiring the LLM to retain awareness of prior actions and dialogue context. To avoid excessive context growth and degraded model performance, we implement a selective history management mechanism that appends only semantically essential dialogue content while discarding auxiliary prompt information. This design preserves core contextual semantics with significantly reduced input length, enabling longer interaction horizons. Dialogue entries are further annotated with role identifiers to ensure compatibility with standard LLM APIs.

2. Planning and execution algorithm: Following the ReAct paradigm, the agent iteratively performs Think-Act-Observe cycles to complete tasks, interacting with the Revit environment and using execution feedback to guide subsequent actions. The algorithm also determines appropriate task termination and governs the overall interaction workflow, as detailed in Section 3.2.
3. Executor: Parsed agent commands are converted into a structured JSON format and passed to the executor for interface invocation. Serving as an intermediary between the agent core and the Revit interface library, the executor parses inputs, calls the corresponding Revit Interfaces, and returns execution results to the agent core as part of the dialogue.
4. Prompt library: To guide LLM reasoning and execution, two categories of prompts are designed, each combining core instructions, response formats, examples, and interface information. The prompt library is detailed in Section 3.3.
5. LLM APIs: Due to the high resource cost of local deployment, this study adopts an API-based approach to access remote LLMs. The OpenAI API is used for its compatibility with the C# environment in Revit, enabling prompt submission and response retrieval within the agent workflow.

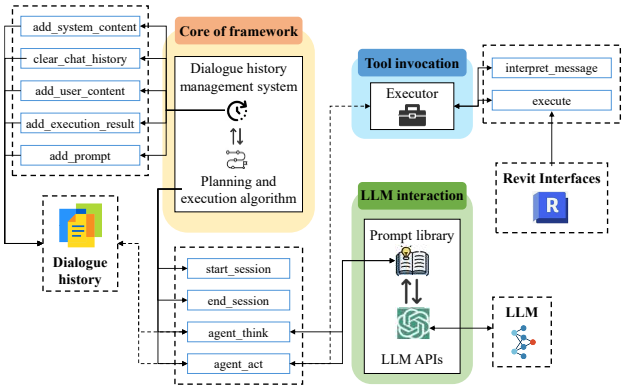

Figure 4. Interaction among the main agent components

3.2 Core of framework Design Based on ReAct

The operation of the agent requires coordinated management of multiple components and proper handling of user-agent dialogue logic. To this end, a ReAct-based core of framework is designed to govern the overall interaction workflow of the Revit Agent, which is executed at the start of each user session. The algorithm is organized into three main stages: initialization, dialogue, and termination, with execution actions conditionally triggered during the dialogue stage, as shown in Figure 5.

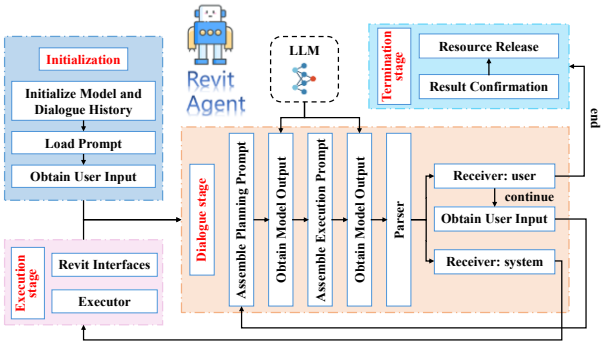

Figure 5. Core of framework design based on ReAct

During initialization, the LLM instance, dialogue history manager, and prompt library are initialized, and users may select different LLM backbones supported by the OpenAI API through configuration settings. The dialogue history manager creates a new history container and populates it with system prompts and synthetic agent-user exchanges that simulate background inquiry, preparing the agent for subsequent multi-turn interaction. Interface-related information is embedded into the dialogue history to support later tool invocation. Finally, the user provides a task description, which serves as the primary objective for the subsequent interaction process.

During the dialogue stage, the algorithm guides the agent through multi-turn interaction following the ReAct Think-Act-Observe paradigm. In each turn, a Think Prompt from the prompt library is first combined with the dialogue history to generate a new prompt, and the LLM produces an unstructured response to plan subsequent actions. Then an Act Prompt that enforces a structured JSON output defined as a Message. The generated JSON is parsed into a C# Message object, enabling executable control flow, as illustrated in Figure 6. Based on the Receiver field in the message, two execution paths are defined. If the receiver is the user, the algorithm evaluates the End flag to determine whether to terminate the interaction or proceed to the next dialogue turn after receiving user input. If the receiver is the system, the

algorithm enters the execution stage, where the executor parses the Call and Args fields to invoke the corresponding Revit Interface. The execution result is then appended to the dialogue history, and the algorithm proceeds to the next interaction cycle.

```
{
"Receiver": system
"Call": RetrieveElement
"Args":{
        "elementIdStrings": null,
        "builtInCategoryName": "OST_Walls",
        "className": null,
        "familyIdString": null,
        "familySymbolIdString": null,
        "levelIdString": null,
       "isElementType": false,
       "parameterFilterArgs": null
   }
}
```

Figure 6. Example of a message class object

The termination stage indicates that the agent determines the task to be completed or infeasible. All executed actions are subject to user confirmation, and any rejected actions are rolled back to ensure BIM model data integrity. Finally, all allocated resources are released, and the dialogue session and task execution are concluded.

3.3 Prompt Design with Multi-Information Integration

LLM outputs are highly sensitive to prompts, making prompt engineering essential. To ensure correct understanding of Revit-specific context and appropriate action generation, instructions, Revit background information, and interface descriptions are integrated into structured prompts. These prompts are organized into three types: Think Prompts, Act Prompts, and Interface Information Prompts.

1. Think Prompts: Consist of core instructions and response examples. The core instructions guide the agent to plan subsequent actions based on the dialogue history, while the examples illustrate reasonable planning responses.
2. Act Prompts: Include core instructions, response examples, and summarized interface information. The core instructions require the agent to generate a structured Message according to a predefined format. While the schema constrains the output format, the examples illustrate correct interface selection and parameter usage. The interface summary briefly lists available interfaces and their functionalities to mitigate LLM forgetting.
3. Interface Information Prompts: Provide complete interface descriptions, including interface names, functionalities, and input-output specifications. Due to their length, these prompts are only injected during the initialization stage.

```
{
"Name": Think Prompt
"Instruction": Based on the task requirements I've presented, and taking into account the historical conversation information, please think in an organized way about what should be accomplished next, what Revit interfaces should be called, or what necessary information you need me to add. If the task has been completed, please think about how you should report and summarize the results to me. Please note that what you think about at this stage will not be executed, you can simply reply in natural language.
"Example": The user wants to get the IDs of all wall instances and the user has given the built-in category ID of the wall as \"OST_Walls\". Among the interfaces provided, the \"RetrieveElement\" interface can filter the eligible element instances by the built-in category. So next, we can call this interface and set the input parameter \"builtInCategoryName\" to \"OST_Walls\", set \"isElementType\" to false and the remaining parameters to null.
}
```

```
{
"Name": Act Prompt
"Instruction": Based on the task requirements and the contents of your thinking above, please reply in JSON format. You can choose to reply to the Revit system to call a specific Revit interface to continue completing the task, or reply to me to ask for more necessary information or report the results after the task is completed. The two types of replies have different format requirements.
"Example": {
        \n Receiver = "system", \n Call = "RetrieveElement", \n Args = {\n { "elementIdStrings", null }, \n
        { "builtInCategoryName", "OST_Walls" }, \n { "className", null }, \n { "familyIdString", null }, \n
        { "familySymbolIdString", null }, \n { "levelIdString", null }, \n { "isElementType", false }, \n { "parameterFilterArgs", null \n }}
   }
}
```

Figure 7. Prompt examples

4 Benchmark for Agent-Based BIM Design Assistance Tasks

4.1 Manually Annotated Test Dataset

Due to the lack of publicly available datasets for Revit-based BIM design tasks, the benchmark dataset is manually collected and annotated to ensure reliability and evaluation relevance. All tasks supported by the current interfaces are categorized into four types: information querying, model visualization, component modification, and component deletion. Tasks are designed for each category based on a reference BIM model and real-world design scenarios, with multiple difficulty levels introduced according to operational complexity.

Each task is accompanied by a manually written reference answer describing the correct execution outcome or feasible interface invocation. In total, 100 English test tasks with annotated answers are constructed to ensure balanced evaluation across different LLMs, with task statistics shown in Table 3.

Table 3. Task statistics

Task Categories	Quantity
Information querying	52
Model visualization	18
Component modification	20
Component deletion	10

4.2 Evaluation Criteria

In practical BIM design and visualization tasks, users primarily focus on the correctness of final results rather than execution details. Accordingly, this study adopts an outcome-oriented evaluation strategy using two metrics: task completion rate and task accuracy.

The task completion rate measures whether the agent successfully completes a task and reports the result, while task accuracy evaluates whether the completed task produces correct outcomes consistent with user requirements. Both metrics are computed as the ratios of completed or accurate tasks to the total number of tasks. Evaluation is conducted by manually comparing the agent's outputs and execution results against annotated reference answers.

5 Experiment

To assess the reliability of the proposed agent system and the impact of different LLM capabilities, three representative models from the GLM series, GLM4-flash, GLM4-plus, and GLM-Zero-Preview, are evaluated. These models respectively represent mid-level general-purpose performance, flagship general-purpose capability, and reasoning-oriented models with strong logical inference ability.

To facilitate a more detailed analysis of LLM performance, results are reported by task category and difficulty level in addition to overall performance. Tasks are grouped into information querying, model visualization, component modification, and component deletion, while difficulty is defined by the minimum number of interaction turns required for completion ($\leqslant$2, 3-4, and $\geqslant$5 turns for Levels 1-3, respectively). This criterion reflects increasing reasoning complexity and dialogue history length.

5.1 Overall Results

Table 4 summarizes the results on the test dataset. The results show that GLM4-plus achieves the best performance, reaching 90% accuracy, which exceeds prior LLM-assisted BIM methods such as BIMS-GPT [16] under zero-shot prompting. It should be noted that, due to differences in tasks and datasets, the comparison with prior studies should be interpreted as a qualitative reference rather than a strictly equivalent benchmark. GLM-Zero-Preview and GLM4-flash show similar accuracy but lower performance than GLM4-plus, while GLM-Zero-Preview exhibits a notably lower completion rate despite high correctness on completed tasks. This is mainly due to its tendency to request additional information when handling complex interface calls. Overall, the results indicate that GLM4-plus enables more reliable BIM-assisted design and visualization within the proposed agent framework.

Table 4. Overall results

Model	Context Limit (K)	Completion rate (%)	Accuracy (%)
GLM4-flash	128	70	93
GLM4-plus	128	90	99
GLM-Zero-Preview	16	71	71

5.2 Performance Across Task Categories

Across task categories, the three LLMs exhibit distinct strengths, as shown in Figure 8-9. GLM4-plus achieves near-perfect completion and 100% accuracy in component modification and deletion, with consistently high performance across all tasks. GLM4-flash performs well in information querying and component deletion but is less effective in model visualization and component modification. GLM-Zero-Preview excels in component modification, while its identical completion and accuracy rates indicate that all completed tasks are correct and all failures result from incomplete execution, mainly due to incorrect judgments about required task information rather than interface invocation errors.

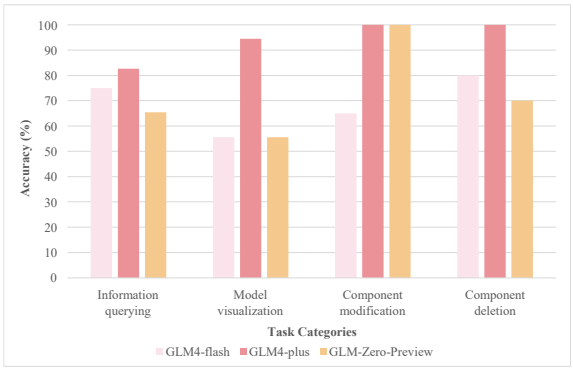

Figure 8. Classification accuracy

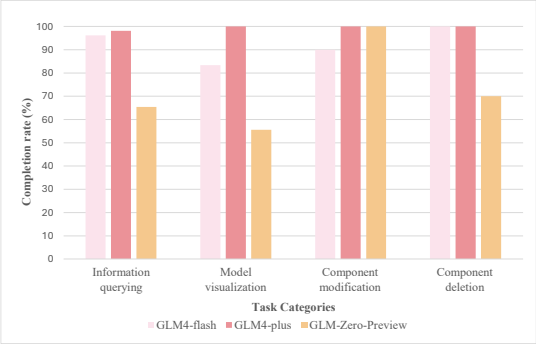

Figure 9. Completion rate

5.3 Performance Across Task Difficulty Levels

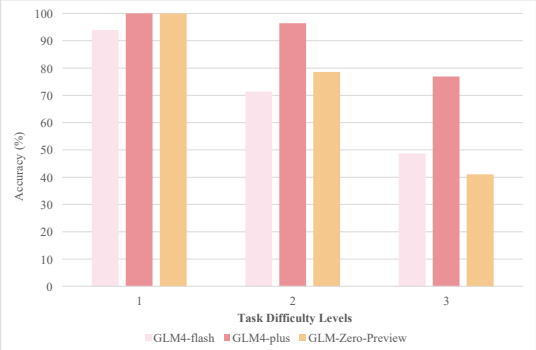

Figure 10. Classification accuracy

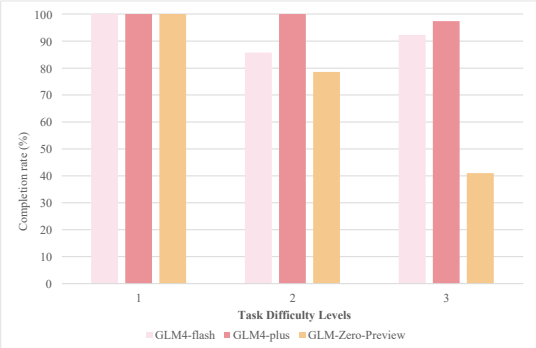

Figure 11. Completion rate

The benchmark includes 33 Level-1, 28 Level-2, and 39 Level-3 tasks, yielding a balanced difficulty distribution. As expected, both accuracy and completion rate decrease with increasing difficulty for all models. While all LLMs perform well on Level-1 tasks, accuracy for GLM4-flash and GLM-Zero-Preview drops sharply on Level-2 and Level-3 tasks, reflecting increased reasoning complexity with longer interactions.

Completion rate analysis further shows that GLM4-flash behaves more aggressively by attempting task completion at all difficulty levels regardless of correctness, while GLM-Zero-Preview adopts a more conservative strategy, frequently requesting additional user input when confidence is insufficient, leading to a higher rate of incomplete tasks.

Figure 12 presents an end-to-end example of a GLM4-plus–based agent successfully completing a task in the model visualization category (difficulty level 2). The task is to display all door instance elements on Level 2 in the Revit 3D view. As shown, the agent first retrieves the corresponding elements using the RetrieveElement interface and then invokes ShowElements based on the returned element IDs. By sequentially using system responses to guide subsequent interface calls, the agent accurately completes the task.

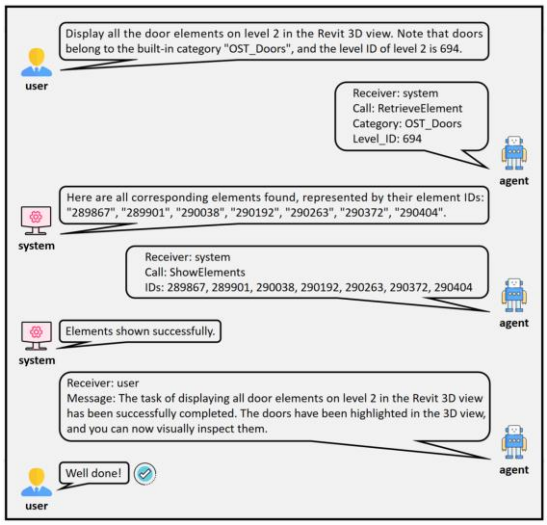

Figure 12. An end-to-end example

6 Conclusion

This study presents a BIM design assistant driven by LLM agents and implements a reliable prototype within the Autodesk Revit environment The main contributions are as follows:

1. A two-layer interface architecture is designed and implemented, accompanied by structured interface descriptions. This design enables real-time manipulation of Revit BIM models by the agent and ensures efficient and accurate interface invocation for seamless BIM interaction.
2. A BIM design agent is developed based on ReAct and Prompt Engineering. By integrating dialogue history management, ReAct-based reasoning, and structured prompt engineering, the agent is endowed with step-by-step reasoning capability, allowing it to handle complex tasks and long-

horizon interactions effectively.

3. An agent-oriented BIM-assisted design benchmark is constructed, introducing task completion rate and task accuracy as evaluation metrics. A total of 100 manually annotated test tasks are created to systematically evaluate the reliability and accuracy of the proposed system.

Experimental results show that high-performance general-purpose LLMs, represented by GLM4-plus, can reliably act as BIM design assistants within the proposed framework, accurately completing tasks of varying types and difficulty levels with an accuracy of up to 90%. These findings demonstrate that, when combined with appropriate agent architectures and domain-specific interfaces, LLM-driven design assistants can provide reliable and practical support for BIM-assisted design and visualization.

Acknowledgments

The authors are grateful for the financial support received from the National Natural Science Foundation of China (No. 52378306).

References

[1] Hu Z Z, Leng S, Lin J R, et al. Knowledge extraction and discovery based on BIM: a critical review and future directions. Arch Computat Methods Eng, 2022; 29(1): 335-356. https://doi.org/10.1007/s11831-021-09576-9.

[2] Migilinskas D, Popov V, Juocevicius V, et al. The Benefits, Obstacles and Problems of Practical Bim Implementation. Procedia Eng 2013;57:767–74. https://doi.org/10.1016/j.proeng.2013.04.097.

[3] Han J, Lu X, Lin J. Pretrained graph neural network for embedding semantic, spatial, and topological data in building information models. Comput-Aided Civ Infrastruct Eng 2025;40:4607–31. https://doi.org/10.1111/mice.70073.

[4] Pan, Y., Wang, M., Lu, L., Lamsal, R., Pärn, E., Zlatanova, S., & Brilakis, I. (2026). LLM-enabled multi-agent framework for natural language interaction with graph-based digital twins. *Automation in Construction*, *183*, 106791.

[5] Huang P-H, Song S-Y, Xu Z, et al. Intelligent BIM Searching via Deep Embedding of Geometric, Semantic, and Topological Features. Buildings 2025;15:951. https://doi.org/10.3390/buildings15060951.

[6] Ni X-R, Pan P, Lin J-R. What makes a good BIM design? Quantifying the link between design behavior and quality. Autom Constr 2025;171:105992. https://doi.org/10.1016/j.autcon.2025.105992.

[7] Zhou, Y. C., Zheng, Z., Lin, J. R., & Lu, X. Z. (2022). Integrating NLP and context-free grammar for complex rule interpretation towards automated compliance checking. Computers in Industry, 142, 103746.

[8] Zheng Z, Han J, Chen K-Y, et al. Translating regulatory clauses into executable codes for building design checking via large language model driven function matching and composing. Eng Appl Artif Intell 2026;163:112823. https://doi.org/10.1016/j.engappai.2025.112823.

[9] Zheng, Z., Lu, X. Z., Chen, K. Y., Zhou, Y. C., & Lin, J. R. (2022). Pretrained domain-specific language model for natural language processing tasks in the AEC domain. Computers in Industry, 142, 103733.

[10] Lin T-H, Huang Y-H, Putranto A. Intelligent question and answer system for building information modeling and artificial intelligence of things based on the bidirectional encoder representations from transformers model. Autom Constr 2022;142:104483. https://doi.org/10.1016/j.autcon.2022.104483.

[11] Jiang G, Ma Z, Zhang L, et al. EPlus-LLM: A large language model-based computing platform for automated building energy modeling. Appl Energy 2024;367:123431. https://doi.org/10.1016/j.apenergy.2024.123431.

[12] Nakano R, Hilton J, Balaji S, et al. WebGPT: Browser-assisted question-answering with human feedback 2022. https://doi.org/10.48550/arXiv.2112.09332.

[13] Brown TB, Mann B, Ryder N, et al. Language Models are Few-Shot Learners. Adv. Neural Inf. Process. Syst. 33, 2020.

[14] Ying H, Sacks R. From Automatic to Autonomous: A Large Language Model- driven Approach for Generic Building Compliance Checking. Proc. CIB W78 Conf., 2024.

[15] Jang S, Lee G, Oh J, et al. Automated detailing of exterior walls using NADIA: Natural-language-based architectural detailing through interaction with AI. Adv Eng Inform 2024;61:102532. https://doi.org/10.1016/j.aei.2024.102532.

[16] Zheng J, Fischer M. BIM-GPT: a Prompt-Based Virtual Assistant Framework for BIM Information Retrieval 2023. https://doi.org/10.48550/arxiv.2304.09333.