BIMgent: Towards Autonomous Building Modeling via Computer-use Agents

Zihan Deng [1 2] **Changyu Du** [1 2] **Stavros Nousias** [1 2] **André Borrmann** [1 2]
https://tumcms.github.io/BIMgent.github.io/

Abstract

Existing computer-use agents primarily focus on general-purpose desktop automation tasks, with limited exploration of their application in highly specialized domains. In particular, the 3D building modeling process in the Architecture, Engineering, and Construction (AEC) sector involves open-ended design tasks and complex interaction patterns within Building Information Modeling (BIM) authoring software, which has yet to be thoroughly addressed by current studies. In this paper, we propose **BIMgent**, an agentic framework powered by multimodal large language models (LLMs), designed to enable autonomous building model authoring via graphical user interface (GUI) operations. BIMgent automates the architectural building modeling process, including multimodal input for conceptual design, planning of software-specific workflows, and efficient execution of the authoring GUI actions. We evaluate BIMgent on real-world building modeling tasks, including both text-based conceptual design generation and reconstruction from existing building design. The design quality achieved by BIMgent was found to be reasonable. Its operations achieved a 32% success rate, whereas all baseline models failed to complete the tasks (0% success rate). Results demonstrate that BIMgent effectively reduces manual workload while preserving design intent, highlighting its potential for practical deployment in real-world architectural modeling scenarios. Code available at: https://github.com/ZihanDDD/BIMgent

[1]Chair of Computing in Civil and Building Engineering, Technical University of Munich, Germany [2]TUM Georg Nemetschek Institute, Munich, Germany. Correspondence to: Zihan Deng <zihan.deng@tum.de>, Changyu Du <changyu.du@tum.de>.

Workshop on Computer-use Agents @ ICML 2025, Vancouver, Canada. *Copyright 2025 by the author(s).*

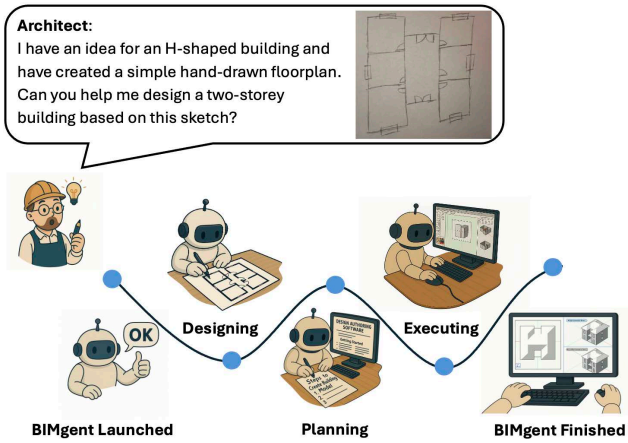


Figure 1. The **BIMgent** framework, enabling the architectural building modeling process to be performed autonomously through computer control.

1. Introduction

To achieve generality in current autonomous agents, researchers are exploring computer-use agents that operate directly through graphical user interfaces (GUIs) (Anthropic, 2024; OpenAI, 2025c). These agents perceive the same screens as humans, generate plans based on the current GUI state, and produce keyboard and mouse actions to perform tasks autonomously (Zhang et al., 2025a). Today, many computer-use agents are being developed to automate tasks across a variety of environments, including the web (Zheng et al., 2024; Zhang et al., 2025b; Zheng et al., 2025a), mobile devices (Wu et al., 2025; Zheng et al., 2025b), and video games (Raad et al., 2024; Tan et al., 2024b).

However, computer-use agents are still underexplored in the Architecture, Engineering, and Construction (AEC) sector. In recent years, Building Information Modeling (BIM) has become indispensable. A BIM model is a digital representation that captures not only the 3D geometry of a building but also includes rich semantic and topological information, enabling support throughout the building's entire lifecycle (Borrmann et al., 2018). Before actual construction begins, architects and engineers typically design and create BIM models using professional BIM authoring software (Baduge et al., 2022). However, there are two main challenges in

the modeling process using such software. First, the commands and GUI of design software are often highly complex, resulting in a steep learning curve and high training costs (Hossain & Zaman, 2022). Second, the design and modeling workflow involves numerous repetitive operations, which further increases the manual effort and time consumption (Heaton et al., 2019).

Computer-use agents can automate the building modeling process by interacting directly with the software GUI, replacing cumbersome manual operations (Agashe et al., 2024). Despite extensive research on computer-use agents in other domains, their application in highly specialized building designs poses unique challenges: (1) Agents must understand the conceptual intent of human design and accurately translate it into command flows within the BIM authoring software for 3D building modeling. (2) BIM authoring software exposes highly parameterized operations and multiple interaction modes. Agents need robust strategies to navigate these options. (3) The richly detailed, noise-filled GUI of design software can overwhelm vision-based agents. Filtering out irrelevant elements while retaining essential visual cues is crucial for dependable operation. (4) Modeling a building involves hundreds of interdependent operations. Agents must not only plan these steps efficiently but also manage error recovery and state tracking across a lengthy workflow.

In response, we introduce **BIMgent**, an agentic framework designed for the autonomous architectural building modeling process, as illustrated in Figure 1. BIMgent is capable of transforming multimodal design intents, including textual building descriptions or rough 2D floorplan sketches, into a 3D BIM model. It not only goes beyond basic interface-level tasks to handle open-ended design generation but also incorporates domain-specific knowledge to plan and execute modeling tasks within complex software environments.

To enhance the GUI agent's capability to operate the design authoring software, we propose a hierarchical planning structure. A high-level planner is designed for generating general design steps. It decomposes the overall modeling workflow into element-level steps (e.g., design layers, walls, etc.). Each step is then passed to a low-level planner, which retrieves relevant information from official software documentation to learn the usage of tools and commands, and generate appropriate GUI actions. We design two distinct workflows for action execution. The *Pure-Action Workflow* handles tasks such as keyboard shortcuts and element placement actions, which are common usage patterns in design software. GUI pixel-level coordinates required for keyboard and mouse operations are either mapped from the floorplan image or retrieved from the software documentation. Inspired by speculative multi-action execution in UFO-2 (Zhang et al., 2025b), we pre-generate the entire action sequence during the planning phase, rather than issuing one action at a time through repeated agent calls during execution to reduce latency. For the *Vision-Driven Workflow* that requires fine-grained information comprehension in GUI, we propose a dynamic GUI grounding method that reduces visual noise by restricting the grounding process to regions associated with relevant interactions for action generation. Considering that the modeling tasks often require hundreds of sequential actions, where a single mistake can lead to cascading errors, we use LLMs as judges to evaluate each action step and provide real-time feedback to correct mistakes, enabling BIMgent to self-reflect.

Existing benchmarks for GUI agents primarily focus on web, mobile, or office software (Bonatti et al., 2024; Xie et al., 2024). To systematically assess the BIMgent, we design a Mini Building Benchmark tailored to assess GUI agent performance in building modeling scenarios. It includes 25 real-world building modeling tasks, which evaluate the agent's ability to handle open-ended design requirements (design evaluation) and performance throughout the modeling process within the BIM authoring software (operation evaluation). The experimental results show that BIMgent achieves an average score of 3 out of 5 across six critical design evaluation criteria. In terms of operation, it achieves a 32% end-to-end success rate across 25 design tasks. Notably, when decomposing the building modeling tasks, it achieves success rates of 86.58% and 95.12% in creating repetitive and redundant elements: walls and openings respectively. By contrast, the strongest baseline model finished none of the end-to-end tasks (0%) and achieves only 31.70% and 35.36% success on walls and openings respectively in our Mini Building Benchmark. These results demonstrate its potential to significantly reduce human effort in the building modeling process.

2. Related Work

Generative AI in AEC. The advent of generative AI has transformed the architecture, engineering, and construction (AEC) domain. Luo and Huang (2022) proposed FloorplanGAN, which integrates vector-based generation with raster-based discrimination for architectural floorplan generation. As the field shifts toward 3D design, Ennemoser and Mayrhofer-Hufnagl (2023) introduced a 3DGAN model that reconstructs architectural forms through a voxel-to-image-to-voxel pipeline; however, their approach emphasizes geometry and lacks semantic detail. Addressing this gap, Gao et al. (2024) developed DiffCAD, a weakly supervised probabilistic model that retrieves and aligns CAD models from RGB images. They essentially have the ability to handle open-ended tasks and fulfill building design requirements. However, existing methods still lack the capability to accept multimodal inputs for handling design changes.

Multimodal LLM Agents. Recent multimodal LLM agents have demonstrated strong capabilities in interacting with software environments, even when addressing open-ended design tasks. For example, in video game environments, Voyager leveraged LLMs to autonomously explore and acquire diverse skills in Minecraft, though it relied on internal APIs for action execution (Wang et al., 2023). TWOSOME combined LLMs with reinforcement learning to improve decision-making in complex scenarios (Tan et al., 2024a). In the domain of 3D scene generation, Hu et al. (2024) translated natural language into 3D environments by implementing an LLM agent that generates Python code. Similarly, Du et al. (2024a) used LLMs to create multiple early-stage building models through internal APIs. However, these methods face limitations due to their reliance on software-specific APIs, which restrict generalizability across different platforms and tools.

GUI Agents. To address the limitations posed by API restrictions, recent GUI agents have demonstrated strong proficiency in interface manipulation. State-of-the-art systems such as UFO-2 (Ren et al., 2020), AgentS2 (Agashe et al., 2025), SEEACT (Zheng et al., 2024), and FRIDAY (Wu et al., 2024) have achieved high performance on GUI-based tasks. Their integration of strategies such as knowledge retrieval and hierarchical planning further enhances their efficiency and ability to handle complex, multi-step tasks across web environments. Similarly, Anthropic (Anthropic, 2024) and OpenAI (OpenAI, 2025c) have developed screenshot-driven agents that automate operations using visual input rather than relying on APIs.

However, these systems face limitations when applied to more complex, domain-specific environments. Most existing use cases are centered around web-based interfaces, which tend to be relatively static and less complex. In contrast, adapting to environments like BIM authoring software is significantly more challenging due to complex GUIs and intricate multi-step operations. A comparable situation can be observed in video game environments, which also demand creativity and involve complex usage patterns. For example, SIMA is a GUI agent that interacts via keyboard and mouse across diverse 3D video game environments (Raad et al., 2024). VillagerAgent demonstrated the ability to manage complex task dependencies in large open-ended environments in Minecraft (Dong et al., 2024). Similarly, Tan et al. (2024b) proposed a generalized GUI agent framework Cradle, which is capable of operating across both video games and web applications. Despite these advances, a shared limitation among these agents is the lack of well-defined task completion signals and standardized benchmarks, which hinders consistent evaluation. As a result, many of these works design their own tasks and experimental protocols. Compared to video games, the building modeling process involves less real-time animation but poses its own challenges, such as an open-ended design process, non-intuitive operations, more complex planning and management, and intricate GUIs that are harder to interpret and ground.

3. BIMgent Framework

BIMgent enables an autonomous architectural building modeling process through computer control, spanning from design concept interpretation to final 3D building modeling. As depicted in Figure 2, the framework consists of three key layers: (1) **Design Layer** transforms conceptual or existing designs into 2D floorplans aligned with the GUI coordinate; (2) **Action Planning Layer** generates knowledge-based, hierarchical operation steps based on software documentation; and (3) **Execution Layer** executes actions under agent-based supervision to complete the modeling workflow. The details of each component are explained in the following sections.

3.1. Design Layer

The Design Layer is responsible for transforming a building description or existing design sketches into a 2D floorplan image. In contrast to existing methods that stop at image generation, this layer further extracts necessary design information and maps the image resolution-level coordinates of floorplan elements to the pixel-level coordinates, enabling downstream GUI grounding, planning, and execution.

Floorplan Generation. We explore three approaches to generate 2D floorplan representations from design intent, specifically, (1) we adopt a generative adversarial network (GAN)-based image generation method (Nauata et al., 2021; Fu et al., 2024) to generate floorplans from textual prompts; (2) we leverage LLMs to produce SVG floorplans from textual descriptions or image input; and (3) we employ advanced multimodal LLMs to directly generate floorplan images either from text or conditioned on existing floorplan visuals. Based on empirical comparison, we find that multimodal LLMs demonstrate better capability in translating both abstract design intent and existing visual layouts into coherent and functional 2D floorplans. This enables our framework to support both text-to-building generation and floorplan-to-building transformation. Detailed comparisons of the three approaches are presented in the Appendix F.

Floorplan Segmentation. Despite the strong image generation and understanding capabilities of current LLMs, they still exhibit limitations in recognizing and localizing accurate architectural components within floorplans. To address this, we integrate a floorplan segmentation model to identify and classify architectural elements such as walls and openings in the generated floorplan.

Floorplan Interpretation. We further employ an additional multimodal LLM to proofread and enhance the results, en-

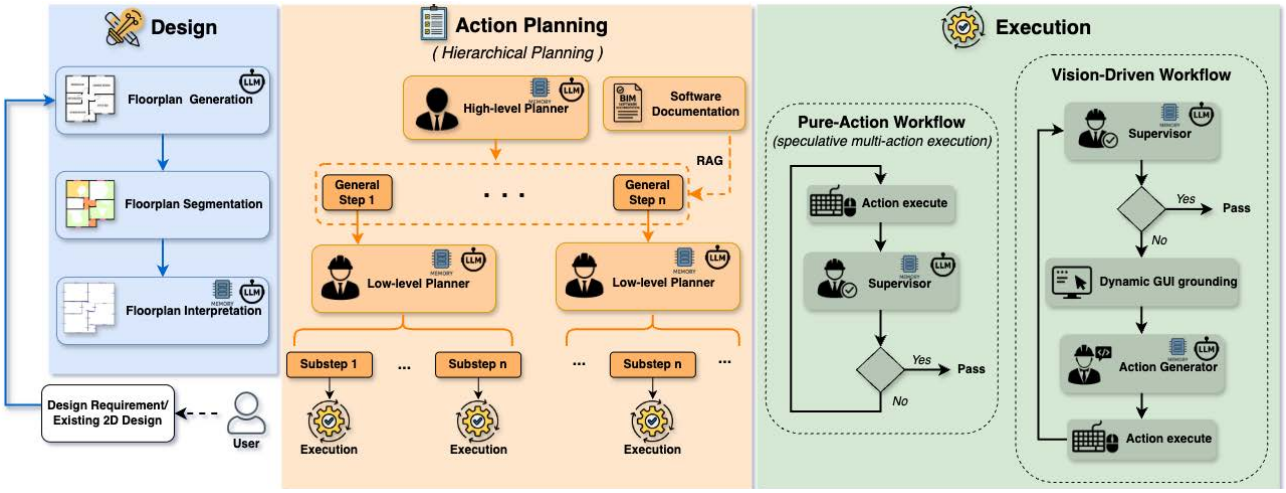


Figure 2. Overview of the BIMgent framework. Given the multimodal design requirements provided by the user, the *Design Layer* first transforms them into a refined floorplan and extracts the necessary semantic and geometric information to guide the modeling process. Based on the interpreted design information and domain knowledge, the *Action Planning Layer* hierarchically organizes the modeling procedure and decomposes it into detailed substeps, guided by the official software documentation. These substeps are then executed through specialized action workflows in the *Execution Layer*, each equipped with verification mechanisms. Execution trajectories are stored in a memory module, which supports both self-reflection and cooperation among different parts of the framework.

abling the extraction of more fine-grained and accurate design details for downstream GUI grounding. For example, it helps distinguish between internal and external walls, or between doors and windows among the openings. Following classification, we apply a rule-based algorithm to map the image-resolution coordinates of the identified components (x_i, y_i) to their corresponding pixel-level coordinates $(x_{\text{gui}}, y_{\text{gui}})$ in the GUI. This mapping can be expressed by the following formula:

$$x_{\text{gui}} = \frac{x_i}{w_{\text{img}}} \cdot w_{\text{gui}}, \quad y_{\text{gui}} = \frac{y_i}{h_{\text{img}}} \cdot h_{\text{gui}}$$

Here, w_{img} and h_{img} denote the width and height of the image resolution, while w_{gui} and h_{gui} refer to the dimensions of the GUI design panel of the BIM authoring software. This scaling is used to convert coordinates from the image space to their corresponding pixel locations in the GUI.

3.2. Action Planning Layer

The entire building modeling process typically involves hundreds of sequential steps. To mitigate errors and improve accuracy, we designed a hierarchical planning process that employs two agents for authoring software action planning: the high-level planner and the low-level planner.

High-Level Planner. After analyzing modeling patterns of several architects, we designed a high-level planner that generates a sequence of general steps based on standard building modeling workflows. For example, the typical process begins with setting up corresponding design layers, followed by creating walls and other elements in a high-level plan. This design imitates the architects' modeling thought process. Additionally, the planner identifies which specific elements from the generated floorplan should be created or configured in each step.

Low-Level Planner. The detailed modeling action steps in the authoring software are complex and involve multiple operation modes, which vary across different users. To handle this complexity, we leverage the official software documentation and adopt a retrieval-augmented generation (RAG) approach (Du et al., 2024b). This allows the agent to explore autonomously and learn software usage dynamically like humans, rather than hard-coding static actions within prompts.

As shown in Figure 3, the documentation is embedded into vector representations and stored in a vector database. Given a general step from the high-level planner, the most relevant sections from the documentation are retrieved. The low-level planner then references both the retrieved documentation and general steps from the high-level planner to generate detailed substeps. It breaks down each general step into multiple actionable substeps, generating them iteratively until the full sequence is completed. We identify two types of substeps generated from the low-level planner, which cover the main usage patterns of human designers interacting with the BIM authoring software: *Vision-Driven* and *Pure-Action*.

Vision-Driven substeps (e.g., Substep 2 in Figure 3) require grounding in the GUI, such as switching tabs or setting

up parameters, which necessitate specific pixel-level coordinates for accurate interactions. These substeps are not directly converted into actions at this stage due to the lack of visual input. In contrast, *Pure-Action* substeps (e.g., Substep 1), such as issuing keyboard shortcuts and placing elements, are deterministic actions, with information typically available in the floorplan metadata or software documentation. The low-level planner directly generates executable actions for Pure-Action tasks without additional GUI grounding. The available actions are detailed in Appendix A.

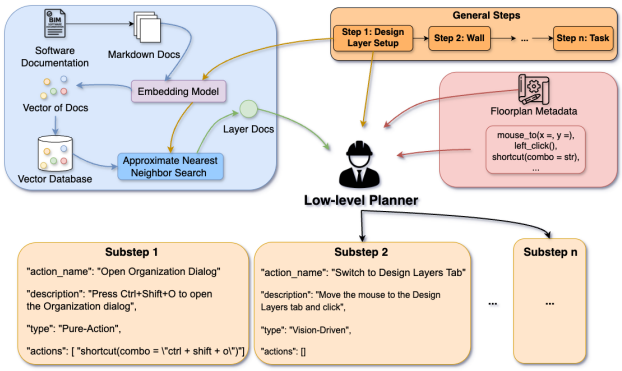


Figure 3. Low-level planner. The general steps generated by the high-level planner are embedded and used to query the official documentation to retrieve the most relevant detailed guidance. Combined with floorplan metadata and the necessary action definitions, these results are forwarded to the agent for detailed subtask generation.

3.3. Execution Layer

The final execution layer implements the planned GUI operations to model buildings within the design software. It sequentially executes the substeps produced by the low-level planner for each general step. Once all substeps for the current general step are completed, it proceeds to the next general step and repeats the process. For the two types of substeps planned by the low-level planner, we accordingly designed dedicated workflows for their execution.

Pure-Action Workflow. Pure-action substeps are executed directly through the planned action sequences. We implement speculative multi-action execution (Zhang et al., 2025b), allowing each substep to be executed without requiring separate API calls for individual actions. After each action is completed, the supervisor is invoked. The supervisor receives the current GUI screenshot and analyzes the result. A common feature of BIM authoring software is that the GUI typically displays metadata about the currently created building elements. Leveraging this, the agent inspects the displayed information to verify whether the created element's type and semantic attributes are correct. If the result is valid, the workflow proceeds to the next task; otherwise, the failed substep is undone and regenerated by the supervisor agent. A detailed visualization of the process is provided in Appendix D.

Vision-Driven Workflow. Vision-driven substeps require dynamic action generation based on the GUI screenshot state. In this workflow, the supervisor agent is invoked first. It captures a GUI screenshot and assesses the current GUI state. If the current state satisfies the substep requirements, the system proceeds to the next substep. If not, we apply a dynamic GUI grounding mechanism for accurate interaction. As shown in Figure 4, to detect visual changes, a screenshot is captured at the beginning of each general step. For example, before starting general step 1, an initial screenshot is taken as a reference. The current GUI screenshot is compared with the initial GUI screenshot to detect visual changes within the differing region, typically a pop-up window. The grounding process then focuses exclusively on this region. This design choice is based on the observation that, in design software, Vision-Driven substeps usually involve parameter settings, which are commonly presented through dedicated pop-up windows. As the surrounding interface remains largely static and irrelevant to the current task, excluding it reduces visual noise. This approach mimics human behavior, where attention is naturally directed toward the changing parts of the interface, resulting in more accurate and efficient grounding. A screen parser model converts screenshots of pop-up windows into structured representations with UI element bounding boxes and text descriptions (i.e., Set-of-Marks). These representations are then passed to a VLM-based action generator, which generates and executes the appropriate actions based on the grounding result. Finally, the supervisor agent re-evaluates the GUI state to verify the success of the operation.

Reflection. In all workflows, a supervisor is integrated to monitor the GUI state and assess whether the current substeps have been successfully completed. In the Vision-Driven Workflow, if a substep fails, the supervisor provides a failure reason to guide future adjustments and support the subsequent Action Generator Agent in regenerating actions. In contrast, in the Pure-Action Workflow, where the required actions are relatively simple, the Supervisor Agent regenerates the actions directly without delegating to another agent.

4. Experiments

4.1. Implementation details

Hybrid Multi-Agent Framework. We employ a hybrid of different LLMs/VLMs as the backbones. We choose OpenAI's gpt-image-1 for floorplan generation due to its advanced image generation and editing capabilities (OpenAI, 2025d). For the interpretation floorplan component, we employ Gemini 2.5 Pro, which demonstrates state-of-

the-art image reasoning capabilities (Google, 2023). For action planning, including the high-level planner, low-level planner, and action generator, we utilize GPT-4.1 to handle more complex reasoning and instruction-following tasks (OpenAI, 2025a). For the supervisor and action generator, we use o4-mini due to its lower cost and faster response time (OpenAI, 2025b).

Non-Agent Components. Our framework incorporates several non-LLM components. For floorplan segmentation, we adopt the DeepFloorPlan (Zeng et al., 2019), a multi-task network trained on about 12k annotated plans that jointly predicts room-boundary primitives (walls and openings) and room-type masks. For software documentation and task embedding, we use OpenAI's text-embedding-3-small (OpenAI, 2024). Finally, during the dynamic GUI grounding process, we employ Omni-Parser-v2 (Yu et al., 2025) for accurate screen parsing and component detection.

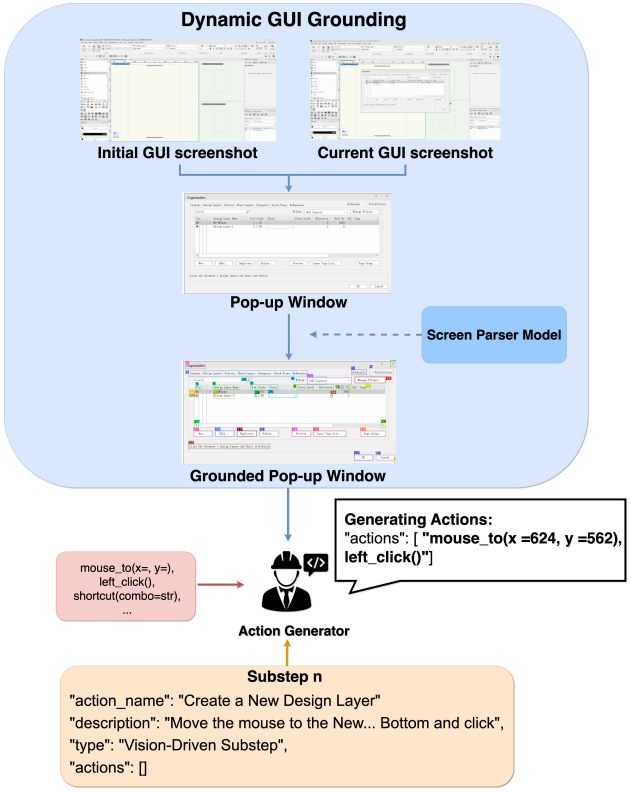


Figure 4. Dynamic GUI Grounding. The initial GUI screenshot is subtracted from the current GUI screenshot to highlight changes, allowing pop-up windows to be specifically visualized and reducing noise during the grounding process. The grounded pop-up window is then passed to the action generator, which combined with the relevant action definitions and substep information, produces the corresponding actions.

4.2. Mini Building Benchmark Introduction

Compared to existing computer-use benchmarks (Zhou et al., 2023; Xie et al., 2024; Deng et al., 2023), where each task typically involves around 10 steps, building modeling tasks are significantly more complex, with each design task usually requiring approximately 100 action steps on average. To evaluate our system, we constructed a custom Mini Building Benchmark consisting of 25 real-world 3D building modeling tasks, all executed within the BIM authoring software Vectorworks. The benchmark includes five tasks for generating 3D buildings from pure textual design requirements, five based on hand-sketched 2D floorplan images, five sourced from the CubiCasa5K floorplan dataset (Kalervo et al., 2019), five from hand-sketched floorplans with additional modification requirements, and five from CubiCasa5K floorplans with modification requirements. In total, the benchmark involves over 2000 action steps. Further details are provided in the Appendix B.

Unlike existing research benchmarks, building modeling tasks lack clear signals for automated evaluation, which makes it difficult to objectively determine whether a task has been successfully completed. To address this, the evaluation process is divided into two phases: **design evaluation** and **operation evaluation**. We conduct human evaluation based on predefined criteria, with assessments performed by architects for design and the final 3D building model. The details are shown in the Appendix C. Because no existing computer-use agents are specifically designed for autonomous building modeling, we choose GPT-4o and Claude 3.7 as baseline models.

5. Results and Analysis

5.1. Experimental Results

Design Evaluation. We asked human architects to grade the generated floorplans based on six design criteria. We then compared our full design layer with two baseline methods: floorplans generated by Claude 3.7 using SVG, and our design layer without the floorplan interpretation module. As illustrated in Figure 5, the results show that our method produces the most reasonable and acceptable designs across all six criteria. Notably, it achieves scores above 3 out of 5 in every aspect, outperforming the baseline models and demonstrating a superior ability to handle open-ended design tasks. The detailed evaluation process can be found in Appendix C.

Operation Evaluation. As shown in Table 1, the BIMgent achieves a 32% end-to-end success rate on the proposed Mini Building Benchmark. The complexity and length of each task make it difficult to directly assess outcomes, as many steps can influence the final success. To enable a more granular evaluation of agent performance, in addi-

Table 1. Success rates (%) on the proposed Mini Building Benchmark test set consisting of 25 building modeling tasks, along with results from the ablation study. N/A indicates 0% success rate.

METHOD	END-TO-END (25)	LAYER (41)	WALL (82)	SLAB (41)	OPENINGS (82)	ROOF (25)
GPT-4O	N/A	N/A	4.87	2.43	12.19	N/A
CLAUDE 3.7	N/A	N/A	31.70	21.95	35.36	N/A
BIMGENT	**32.00**	**46.34**	**86.58**	**73.81**	**95.12**	**60.00**
W/O DYNAMIC GUI GROUNDING	13.33	34.15	86.58	73.81	92.68	38.46
W/O SUPERVISION	11.53	24.19	84.14	70.73	92.68	48.00
W/O HIERARCHICAL PLANNING	N/A	2.43	41.46	58.53	46.34	12.00

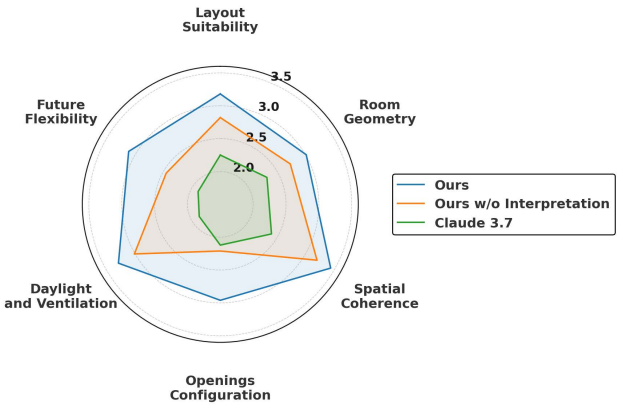


Figure 5. Human evaluation of the generated floorplan designs based on six criteria. Lower values indicate that the corresponding requirements were not clearly specified in the design instructions, while higher values reflect better alignment. The detailed evaluation process can be found in Appendix C.

tion to 25 end-to-end modeling tasks, we also focused on subtasks involving the modeling of key architectural components. Specifically, we evaluate the creation of design layers (41 tasks), walls (82 tasks), slabs (41 tasks), openings (82 tasks), and roofs (25 tasks) based on whether all required architectural elements are successfully created and whether all parameters are properly configured within each subtask. BIMgent performs particularly well on component-related tasks such as wall and opening creation, with 86.58% and 92.68% success rates respectively. This performance is attributable to the availability of floorplan metadata and the relatively simple element creation action patterns that BIMgent can learn effectively. The strongest baseline (Claude 3.7) could not complete any end-to-end modeling task in our benchmark because of the heavy planning and extensive GUI operations that are required. Compared to subtasks, the baseline also performed poorly. BIMgent achieves a 46.34% success rate on the Layer subtask, 60% on Roof (both versus 0% for the baseline), and significantly outperforms the baseline in Wall, Slab, and Openings creation. The results suggest that BIMgent has strong potential to assist in tedious operations by reliably replicating human behavior. Compared to the baseline experiments, our method achieves significantly higher performance, both in overall success rate and across individual modeling parts.

5.2. Ablation Study

Visual Improvement. As illustrated in Table 1, with the proposed dynamic GUI grounding mechanism, the overall success rate increases by 18.67%. Delving into the detailed decomposed components, we find that Layer and Roof, which require extensive grounding for name editing and parameter configuration, show significant improvement with 12.19% and 21.54%, respectively. This targeted visual attention greatly enhances task efficiency and accuracy.

Reflection Ability. The integration of supervision and self-reflection strategies enables the agent to analyze previously failed tasks and generate improved action plans. Statistically, we observe that the overall task success rate increases by 20.57% with this mechanism. The primary reason is that Vision-Driven tasks often fail to generate accurate grounding on the first attempt. Through self-reflection, the performance of tasks such as Layer and Roof was improved by 22.13% and 12% respectively. In addition, the performance of other related components was slightly improved by approximately 3%, primarily due to the correction of rare software errors.

Hierarchical Planning Is Essential. Without hierarchical planning to guide the general building modeling steps, the agent struggles to complete even a single task. Other decomposed tasks also perform poorly, particularly the Vision-Driven subtasks such as Layer and Roof. Given the hundreds of substeps involved and the complex usage patterns of the design software, it is challenging for a single LLM to accurately generate the entire sequence of building actions or to fully comprehend the software's operational logic. The agent exhibits a limited ability to create elements. This demonstrates its learning potential under documentation guidance, though its overall performance remains low.

5.3. Error Analysis

To better visualize BIMgent's performance, we draw inspiration from Agent S (Agashe et al., 2024) and conduct an error rate analysis across all 25 modeling tasks. We trace the BIMgent's trajectory throughout the tasks and analyze all 92 erroneous action steps, categorizing them into three types: (1) Planning Errors – incorrect plans that do not align with the current task; (2) Grounding Errors – failures in accurately identifying or parsing the intended GUI targets; and (3) Execution Errors – incorrect or unintended actions during operation. As shown in Figure 6, our findings reveal that, due to the complexity of the BIM authoring software's GUI and its intricate usage patterns, the most error-prone aspects are those associated with grounding and execution, which account for 40.0% and 45.6% of the total errors, respectively. Detailed visualizations of the three types of errors are presented in Appendix D.

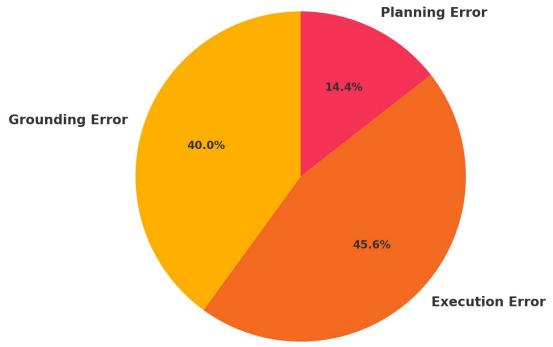


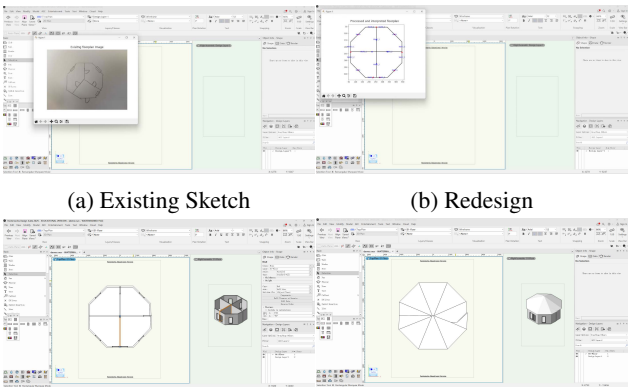
Figure 6. Distribution of the 92 errors across three types: planning errors, grounding errors, and execution errors.

5.4. Qualitative Analysis

In the qualitative analysis, we present a successful example from our benchmark. Given a hand-drawn floorplan and input prompt:

Generate a building model based on a hand-drawn octagon floorplan, modifying the interior layout to include four rooms instead of three.

The creation of a 3D building model is illustrated in Figure 7. The process begins with the system ingesting both the textual description and the hand-drawn image, which it interprets, regenerates, segments, and processes to produce a 2D floorplan. This floorplan is then converted into a structured representation that downstream agents can parse and act upon. In the second stage, the agents use this structured data to sequentially generate actions and construct the model within the BIM software. Finally, a complete 3D building model is generated through automated computer control. More qualitative examples are in Appendix D.



(a) Existing Sketch (b) Redesign

(c) Wall creation (d) Generated 3D building model

Figure 7. Example: *Generate a building model based on a hand-drawn octagon floorplan, modifying the interior layout to include four rooms instead of three.* (a)–(d) show the input, redesigned floorplan, action sequence for wall creation, and final 3D model, respectively. More examples can be found in Appendix D and E.

6. Conclusion

In this paper, we introduce **BIMgent**, an agentic framework powered by multimodal LLMs, capable of autonomously generating building models within BIM authoring software through GUI operations. This paper makes the following key contributions:

(1) Compared to existing GUI agents, **BIMgent** demonstrates the ability to handle open-ended design tasks, bridging the gap in applying GUI agents to professional design software.

(2) **BIMgent** addresses the limitations of domain-specific task handling by integrating software documentation into the planning process through a retrieval-augmented strategy.

(3) **BIMgent** significantly boosts performance by combining dynamic GUI grounding, reflective feedback, and hierarchical planning, thereby overcoming the challenges of complex GUI and the hundreds of steps required for building modeling tasks.

(4) By evaluating two stages of the framework, we show that **BIMgent** can complete the entire modeling process autonomously, particularly excelling in the most labor-intensive parts of the modeling process.

In the future, one primary direction is to extend the framework to other design software, exploring its potential for generalization across platforms. Another key challenge lies in optimizing the agent's step count and execution time. Since our goal is to reduce the substantial manual effort involved in building model authoring, improving efficiency is crucial. Additionally, while we currently rely on existing pre-trained models, fine-tuning an open-source model could

further enhance adaptability and performance. Finally, to address limitations in evaluation, we plan to develop a more automated evaluation method and introduce a dedicated benchmark for more consistent and scalable assessment.

7. Acknowledgment

This work is funded by Nemetschek Group, which is gratefully acknowledged. We sincerely appreciate the licensing support provided by Vectorworks, Inc.

References

Agashe, S., Han, J., Gan, S., Yang, J., Li, A., and Wang, X. E. Agent s: An open agentic framework that uses computers like a human. *arXiv preprint arXiv:2410.08164*, 2024.

Agashe, S., Wong, K., Tu, V., Yang, J., Li, A., and Wang, X. E. Agent s2: A compositional generalist-specialist framework for computer use agents. *arXiv preprint arXiv:2504.00906*, 2025.

Anthropic. Introducing computer use, a new Claude 3.5 Sonnet, and Claude 3.5 Haiku, 2024. URL https://www.anthropic.com/index/claude-3-5-and-tool-use.

Baduge, S. K., Thilakarathna, S., Perera, J. S., Arashpour, M., Sharafi, P., Teodosio, B., Shringi, A., and Mendis, P. Artificial intelligence and smart vision for building and construction 4.0: Machine and deep learning methods and applications. *Automation in Construction*, 141:104440, 2022.

Bonatti, R., Zhao, D., Bonacci, F., Dupont, D., Abdali, S., Li, Y., Lu, Y., Wagle, J., Koishida, K., Bucker, A., et al. Windows agent arena: Evaluating multi-modal os agents at scale. *arXiv preprint arXiv:2409.08264*, 2024.

Borrmann, A., König, M., Koch, C., and Beetz, J. Building information modeling technology foundations and industry practice: Technology foundations and industry practice, 2018.

Deng, X., Gu, Y., Zheng, B., Chen, S., Stevens, S., Wang, B., Sun, H., and Su, Y. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114, 2023.

Dong, Y., Zhu, X., Pan, Z., Zhu, L., and Yang, Y. Villageragent: A graph-based multi-agent framework for coordinating complex task dependencies in minecraft. *arXiv preprint arXiv:2406.05720*, 2024.

Du, C., Esser, S., Nousias, S., and Borrmann, A. Text2bim: Generating building models using a large language model-based multi-agent framework. *arXiv preprint arXiv:2408.08054*, 2024a.

Du, C., Nousias, S., and Borrmann, A. Towards a copilot in BIM authoring tool using large language model based agent for intelligent human-machine interaction. In *Proc. of the 31th Int. Conference on Intelligent Computing in Engineering (EG-ICE)*, Jul 2024b.

Ennemoser, B. and Mayrhofer-Hufnagl, I. Design across multi-scale datasets by developing a novel approach to 3dgans. *International Journal of Architectural Computing*, 21(2):358–373, 2023.

Fu, R., Wen, Z., Liu, Z., and Sridhar, S. Anyhome: Open-vocabulary generation of structured and textured 3d homes. In *European Conference on Computer Vision*, pp. 52–70. Springer, 2024.

Gao, D., Rozenberszki, D., Leutenegger, S., and Dai, A. Diffcad: Weakly-supervised probabilistic cad model retrieval and alignment from an rgb image. *ACM Transactions on Graphics (TOG)*, 43(4):1–15, 2024.

Google, G. T. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Heaton, J., Parlikad, A. K., and Schooling, J. Design and development of bim models to support operations and maintenance. *Computers in industry*, 111:172–186, 2019.

Hossain, S. T. and Zaman, K. U. A. B. Introducing bim in outcome based curriculum in undergraduate program of architecture: Based on students perception and lecture-lab combination. *Social Sciences & Humanities Open*, 6 (1):100301, 2022.

Hu, Z., Iscen, A., Jain, A., Kipf, T., Yue, Y., Ross, D. A., Schmid, C., and Fathi, A. Scenecraft: An llm agent for synthesizing 3d scenes as blender code. In *Forty-first International Conference on Machine Learning*, 2024.

Kalervo, A., Ylioinas, J., Häikiö, M., Karhu, A., and Kannala, J. Cubicasa5k: A dataset and an improved multi-task model for floorplan image analysis. In *Image Analysis: 21st Scandinavian Conference, SCIA 2019, Norrköping, Sweden, June 11–13, 2019, Proceedings 21*, pp. 28–40. Springer, 2019.

Luo, Z. and Huang, W. Floorplangan: Vector residential floorplan adversarial generation. *Automation in construction*, 142:104470, 2022.

Nauata, N., Hosseini, S., Chang, K.-H., Chu, H., Cheng, C.-Y., and Furukawa, Y. House-gan++: Generative adversarial layout refinement network towards intelligent computational agent for professional architects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13632–13641, 2021.

OpenAI. New embedding models and API updates, 2024. URL https://openai.com/index/new-embedding-models-and-api-updates/.

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

OpenAI. Introducing OpenAI o3 and o4-mini, 2025b. URL https://openai.com/index/introducing-o3-and-o4-mini/.

OpenAI. Introducing the computer-using agent, 2025c. URL https://openai.com/index/computer-using-agent/.

OpenAI. Introducing our latest image-generation model in the API, 2025d. URL https://openai.com/index/image-generation-api/.

Raad, M. A., Ahuja, A., Barros, C., Besse, F., Bolt, A., Bolton, A., Brownfield, B., Buttimore, G., Cant, M., Chakera, S., et al. Scaling instructable agents across many simulated worlds. *arXiv preprint arXiv:2404.10179*, 2024.

Ren, Z., Yu, Z., Yang, X., Liu, M.-Y., Schwing, A. G., and Kautz, J. Ufo 2: A unified framework towards omni-supervised object detection. In *European conference on computer vision*, pp. 288–313. Springer, 2020.

Tan, W., Zhang, W., Liu, S., Zheng, L., Wang, X., and An, B. True knowledge comes from practice: Aligning llms with embodied environments via reinforcement learning. *arXiv preprint arXiv:2401.14151*, 2024a.

Tan, W., Zhang, W., Xu, X., Xia, H., Ding, Z., Li, B., Zhou, B., Yue, J., Jiang, J., Li, Y., et al. Cradle: Empowering foundation agents towards general computer control. *arXiv preprint arXiv:2403.03186*, 2024b.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Wu, Q., Liu, W., Luan, J., and Wang, B. Reachagent: Enhancing mobile agent via page reaching and operation. *arXiv preprint arXiv:2502.02955*, 2025.

Wu, Z., Han, C., Ding, Z., Weng, Z., Liu, Z., Yao, S., Yu, T., and Kong, L. Os-copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*, 2024.

Xie, T., Zhang, D., Chen, J., Li, X., Zhao, S., Cao, R., Hua, T. J., Cheng, Z., Shin, D., Lei, F., et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Yu, W., Yang, Z., Wan, J., Song, S., Tang, J., Cheng, W., Liu, Y., and Bai, X. Omniparser v2: Structured-points-of-thought for unified visual text parsing and its generality to multimodal large language models. *arXiv preprint arXiv:2502.16161*, 2025.

Zeng, Z., Li, X., Yu, Y. K., and Fu, C.-W. Deep floor plan recognition using a multi-task network with room-boundary-guided attention. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9096–9104, 2019.

Zhang, C., He, S., Qian, J., Li, B., Li, L., Qin, S., Kang, Y., Ma, M., Liu, G., Lin, Q., et al. Large language model-brained gui agents: A survey. *arXiv preprint arxiv:2411.18279*, 2025a.

Zhang, C., Huang, H., Ni, C., Mu, J., Qin, S., He, S., Wang, L., Yang, F., Zhao, P., Du, C., et al. Ufo2: The desktop agentos. *arXiv preprint arXiv:2504.14603*, 2025b.

Zheng, B., Gou, B., Kil, J., Sun, H., and Su, Y. Gpt-4v (ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*, 2024.

Zheng, B., Fatemi, M. Y., Jin, X., Wang, Z. Z., Gandhi, A., Song, Y., Gu, Y., Srinivasa, J., Liu, G., Neubig, G., et al. Skillweaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025a.

Zheng, J., Wang, L., Yang, F., Zhang, C., Mei, L., Yin, W., Lin, Q., Zhang, D., Rajmohan, S., and Zhang, Q. Vem: Environment-free exploration for training gui agent with value environment model. *arXiv preprint arXiv:2502.18906*, 2025b.

Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

A. BIMgent Action Definitions

We pre-define multiple low-level GUI actions. The action generator and low-level planner responsible for producing actions use these action definitions as references during execution. The detailed information is shown in Table 2.

Table 2. Defined low-level actions including their parameters and descriptions.

ACTION	PARAMETER(S)	DESCRIPTION
`MOVE_MOUSE_TO(X: INT, Y: INT)`	PIXEL COORDINATES	MOVE THE MOUSE CURSOR TO THE SPECIFIED SCREEN POSITION.
`LEFT_CLICK()`	–	PERFORM A LEFT-CLICK USING THE MOUSE.
`TYPE_NAME(NAME: STR)`	NAME STRING	TYPE THE GIVEN NAME USING THE KEYBOARD.
`PRESS_ESCAPE()`	–	PRESS THE ESCAPE KEY ON THE KEYBOARD.
`PRESS_ENTER()`	–	PRESS THE ENTER KEY ON THE KEYBOARD.
`SHORTCUT(COMBO: STR)`	KEY COMBINATION STRING	EXECUTE A KEYBOARD SHORTCUT BY PRESSING THE SPECIFIED KEY COMBINATION.
`SELECT_ALL()`	–	SELECT ALL CURRENT COMPONENTS.

B. Mini Building Benchmark

B.1. Introduction

We present a Mini Building Benchmark consisting of 25 real-world BIM modeling tasks. These tasks cover five input scenarios. Firstly, five text-only conceptual designs that describe key attributes such as the building type, number and types of rooms, and the number of floors (ranging from one to three storeys). Secondly, five hand-drawn sketch floorplan images depict both regular and irregular shapes, along with the number of floors. Thirdly, five randomly unmodified floorplans were selected from the CubiCasa5K dataset (Kalervo et al., 2019), which contains 5000 real estate floorplan images. Fourthly, the same five hand-drawn sketch floorplans were reused with additional explicit modification requirements (e.g., 'add a room'). Fifthly, the same five CubiCasa5K floorplans with similar modification instructions. Figure 8 illustrates the distribution.

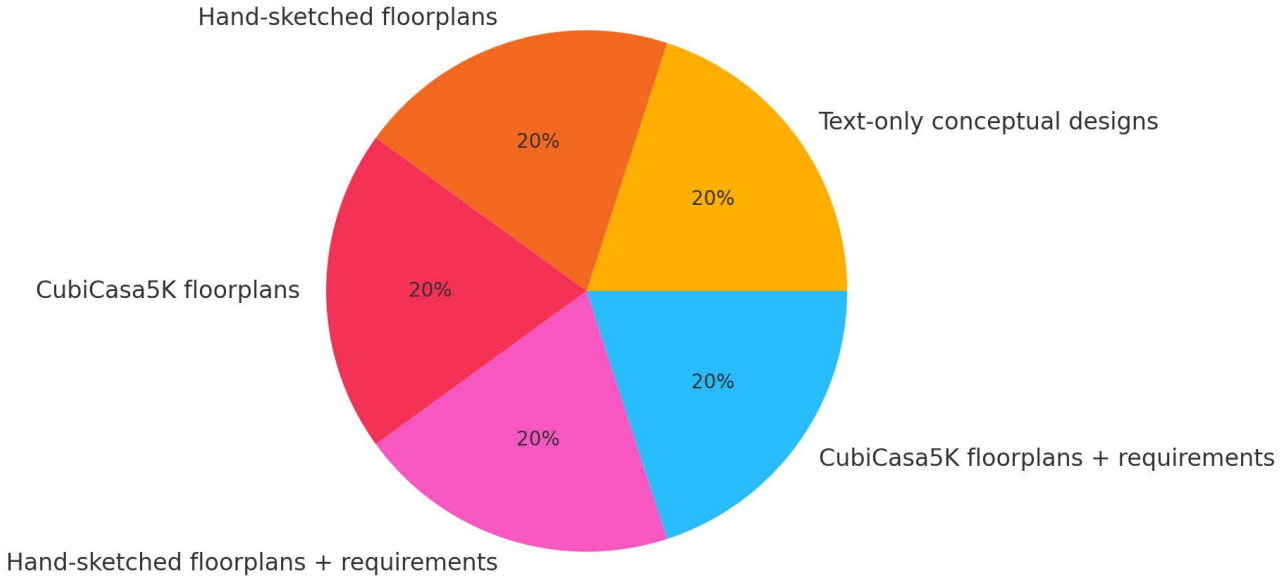


Figure 8. Task Distribution of the Mini Building Benchmark

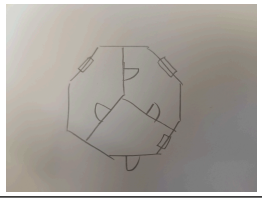
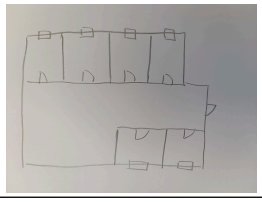
B.2. Tasks

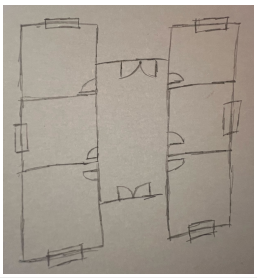
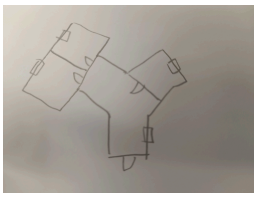
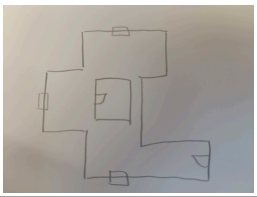
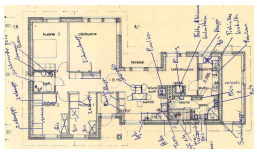
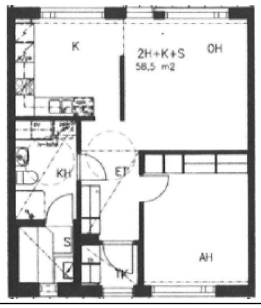
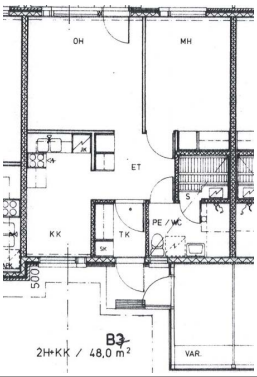
In this section, we present the detailed task contents, as shown in Table 3 and Table 4.

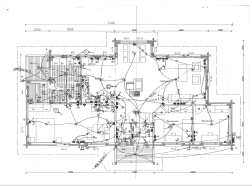
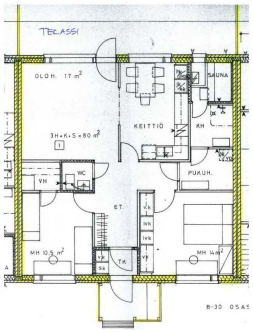
Table 3: Detailed contents of the tasks 1-5 from the Mini Building Benchmark

Task Index	Task content
Task 1	Generate a one-storey office building with a large open workspace occupying most of the floor area. The layout should also include two enclosed meeting rooms, a manager's office, a small pantry, and two restrooms.
Task 2	Generate a one-storey building with a regular hexagonal footprint. Inside the building, create four rooms of roughly equal size. Each room must include doors and one window.
Task 3	Create a two-storey rectangular residential building designed for a single family. The layout must include a living room, kitchen, bathroom, master bedroom, and one additional bedroom.
Task 4	Design a two-storey hospital building with eight distinct rooms. These must include a reception area, two consultation rooms, one minor surgery room, one waiting area, two patient rooms, and one staff room.
Task 5	Create a three-storey commercial office building where each floor has the same layout. Each floor must include two large office rooms, a small meeting room, a restroom, and a central corridor. The entrance to the building is located on the ground floor and leads directly to the corridor.

Table 4: Detailed contents of tasks 6–25 from the Mini Building Benchmark

Task Index	Floorplan Image	Task Content
Task 6 & 16		**Task 6**: Generate a one-storey octagonal building based on the hand-drawn sketch. **Task 16**: Generate a building model based on a hand-drawn octagon floorplan, modifying the interior layout to include four rooms instead of three.
Task 7 & 17		**Task 7**: Generate a two-storey building based on the sketch. **Task 17**: Make a two-floor office building based on the sketch with changes. Split the biggest room in the middle into two rooms.

Task Index	Floorplan Image	Task Content
Task 8 & 18		**Task 8**: Generate a one-storey building based on the sketch. **Task 18**:Make a one-floor building based on the sketch with updates. The building has an H-shape. Add one extra room to both blocks, so each block has four rooms instead of three.
Task 9 & 19		**Task 9**: Generate a two-storey building based on the sketch. **Task 19**: Make a two-floor building based on the sketch with updates. Add one more room to the left wing, so the building has five rooms total.
Task 10 & 20		**Task 10**: Generate a two-storey building based on the sketch. **Task 20**: Generate a two-storey building based on the sketch, remove the small room in the middle.
Task 11 & 21		**Task 11**: Generate a one-storey building based on the image. **Task 21**: Make a one-floor house based on the floorplan image with updates. Add an extra room next to the top left room.
Task 12 & 22		**Task 12**: Generate a one-floor building based on the image. **Task 22**: Make a one-floor apartment based on the image but with updates. Add an additional room in the bottom-right corner.
Task 13 & 23		**Task 13**: Generate a two-floor apartment based on the image. **Task 23**: Generate a two-floor apartment based on the image with updates. Make the left bottom room smaller and add an additional space next to it.

Task Index	Floorplan Image	Task Content
Task 14 & 24		**Task 14**: Generate a two-floor house based on the image. **Task 24**: Generate a two-floor house based on the floorplan image with updates. Add a small room next to the top right room.
Task 15 & 25		**Task 15**: Generate a two-floor building based on the image. **Task 25**: Generate a two-floor house based on the image with some updates. Add a small room in the bottom.

C. Evaluation Criteria

We divided the evaluation process into two phases: design evaluation and operation evaluation. In this section, we introduce the detailed evaluation criteria for these two phases.

C.1. Design Evaluation

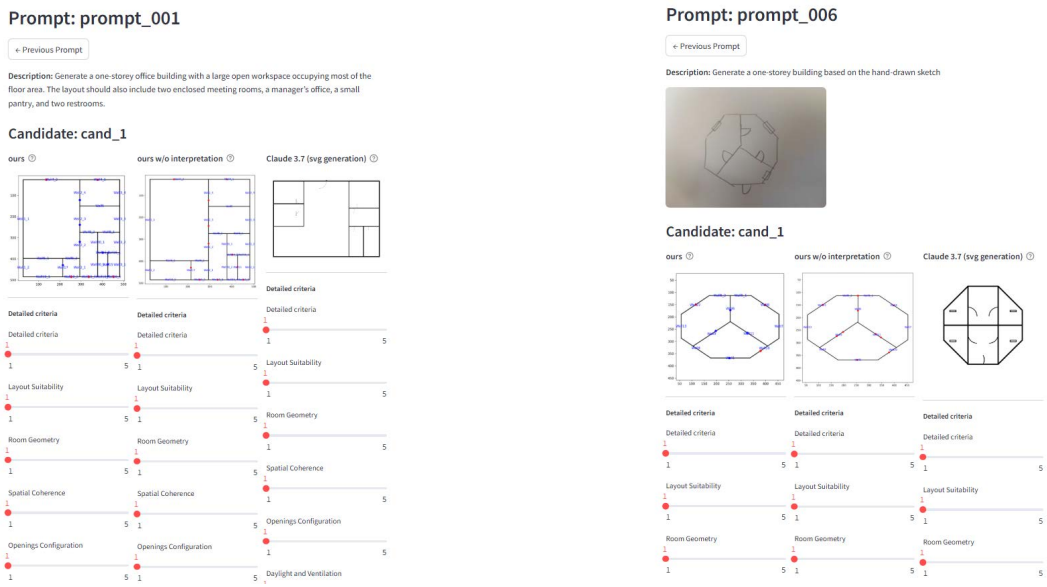
Evaluation Criteria. Since the design task is open-ended, there is no definitive signal to automatically determine whether the design requirements have been fulfilled. Therefore, we rely on human evaluation at this stage. We define six criteria to assess the quality of the generated floorplans, which are shown in Table 5.

Table 5: Detailed descriptions of the criteria for design evaluation

Criteria	Description
Layout Suitability	Evaluate whether the overall layout, including the number and configuration of rooms, meets the specified design requirements.
Room Geometry	Check for geometric validity of rooms, ensuring there are no isolated, distorted, or nonsensical room shapes.
Spatial Coherence	Assess the spatial connectivity between rooms and ensure there are no disconnected or inaccessible areas.
Openings Configuration	Verify that openings such as doors and windows are placed correctly on walls and follow typical architectural conventions.
Daylight and Ventilation	Confirm that each room has access to openings that enable natural light and ventilation.
Future Flexibility	Evaluate whether the floorplan allows for future functional changes, adaptability, or expansions in use.

Evaluation Method We use Claude 3.7 (SVG generation) as the baseline. Additionally, we conduct an ablation study by removing the interpretation part from our design method. To make the evaluation more robust, we expanded the dataset by running the design component of our framework three times for each task in the Mini Building Benchmark, resulting in 75 generated floorplan images per method. For each evaluation criterion, a score of 5 represents full satisfaction of the

requirement. The closer a score is to 5, the better the design fulfills the criterion, and vice versa. We developed a web-based survey platform where the generated images were presented alongside rating questions based on the defined criteria. We invited architects to participate in the evaluation. A snapshot of the survey interface is shown in Figure 9.



(a) Example survey interface for Task 1

(b) Example survey interface for Task 6

Figure 9. User interface for the design evaluation survey.

C.2. Operation Evaluation

For the operation evaluation, we first selected the best-performing design (based on the design evaluation) and treated it as the ground truth reference. For end-to-end operation evaluation, we exported the final output files and calculated the number of each architecture component (e.g., walls, openings, and slabs). These are compared against the ground truth to identify any missing or incorrect elements. We further conducted a human evaluation on each project file. Experts manually inspected the design layer settings, wall heights, roof configurations, and parameter settings to assess correctness and completeness.

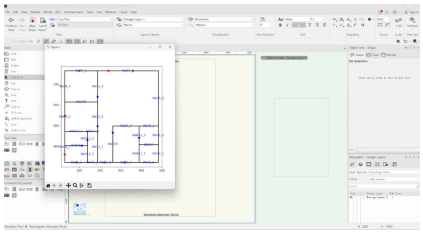
For subtask-level evaluation, we captured screenshots throughout the modeling process, labeling each screenshot according to its corresponding subtask. These screenshots were manually reviewed to identify the specific subtask stage at which any failures occurred. In addition, the final output file was examined to verify whether the component assigned to each subtask was successfully created. If the required component is missing, both the end-to-end task and the corresponding subtask are marked as failed.

D. BIMgent Execution Trajectories on the Mini Building Benchmark

We present the execution trajectories of 2 tasks from the Mini Building Benchmark that were successfully completed by BIMgent, providing a supplementary perspective to the qualitative analysis discussed in Section 5.4. Additionally, we include failure cases referenced in Section 5.3, along with corresponding screenshots and sequences of the generated actions. As tasks involve a large number of action steps, the execution process is broken down into detailed segments.

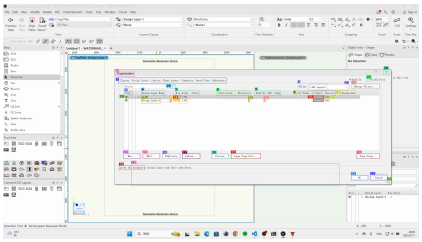
D.1. Task 1 Action Examples

Given the task: *Generate a one-storey office building with a large open workspace occupying most of the floor area. The layout should also include two enclosed meeting rooms, a manager's office, a small pantry, and two restrooms.* The floorplan design and new design layer creation are illustrated in Figure 10, element creation is shown in Figure 11, and roof creation is depicted in Figure 12.



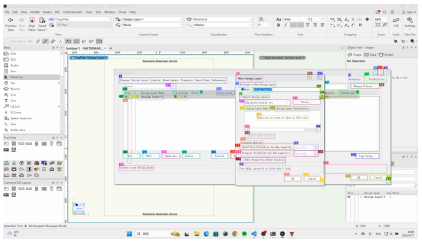
(a)

Floorplan design
–
–



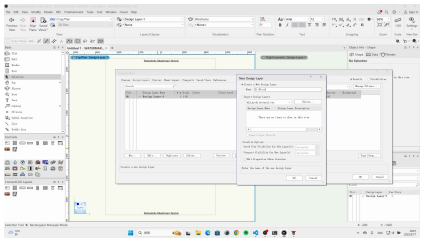
(b)

Open organization dialog
`shortcut(ctrl + shift + O)`
–



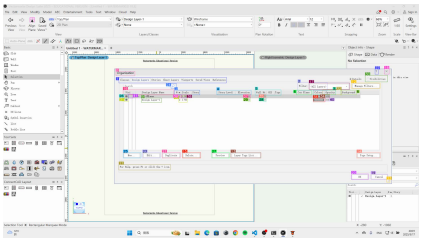
(c)

Click ‘New…’
`move_mouse_to(590,699),`
`left_click()`



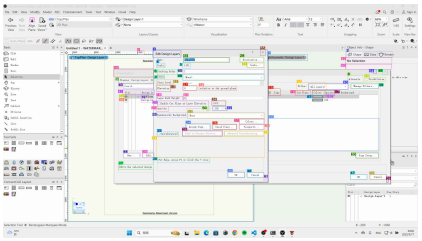
(d)

Type name
`move_mouse_to(1179,396),`
`left_click(),select_all(),`
`type_name("01-Floor")`



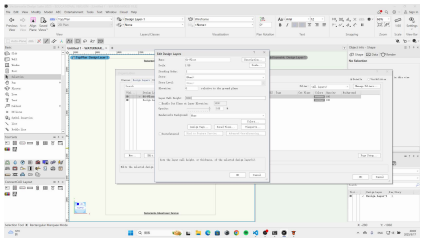
(e)

Confirm
`press_enter()`
–



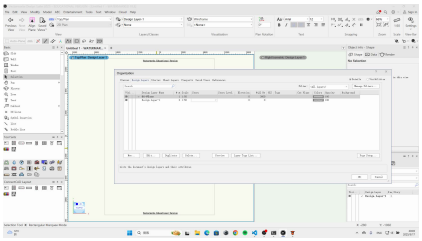
(f)

Click ‘Edit…’
`move_mouse_to(684,698),`
`left_click()`



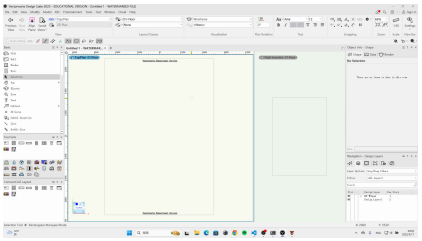
(g)

Edit elevation
`move_mouse_to(870,435),`
`left_click(),`
`select_all(),type_name("3000")`



(h)

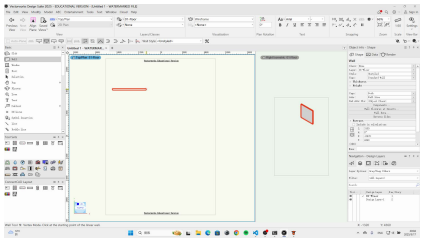
Confirm settings
`press_enter()`
–
–



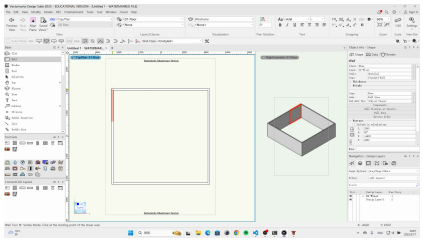
(i)

Final confirmation
`press_enter()`
–
–

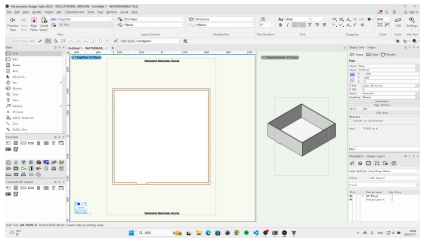
Figure 10. Screenshots of Floorplan Design and Design Layer creation actions.



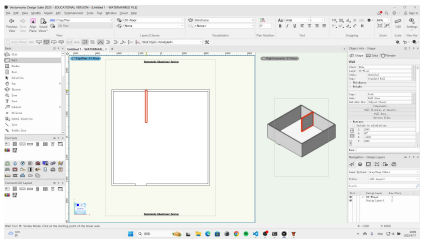
(a)
First external wall
```
shortcut(combo='9')
move_mouse_to(x=500,y=393)
left_click()
move_mouse_to(x=656,y=393)
left_click()
press_enter()
```



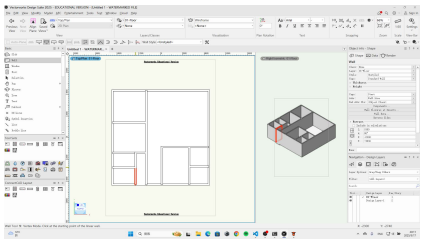
(b)
Last external wall
```
shortcut(combo='9')
move_mouse_to(x=500,y=542)
left_click()
move_mouse_to(x=500,y=393)
left_click()
press_enter()
```



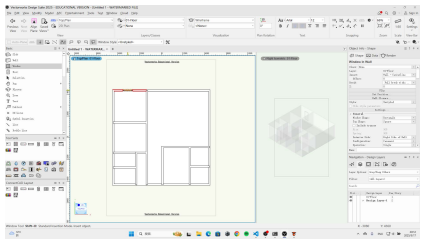
(c)
Create slab by picking external walls
```
shortcut(combo='alt+shift+2')
move_mouse_to(x=578,y=393)
left_click()
...
left_click()
press_enter()
```



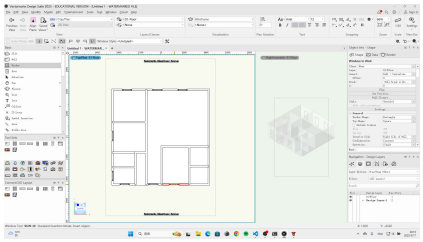
(d)
Create first internal wall
```
shortcut(combo='9')
move_mouse_to(x=656,y=542)
left_click()
move_mouse_to(x=656,y=393)
left_click()
press_enter()
```



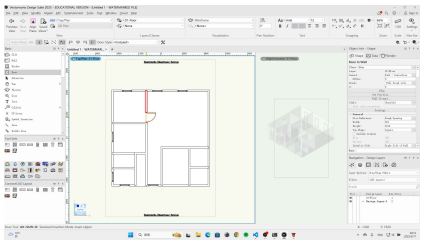
(e)
Final internal wall
```
shortcut(combo='9')
move_mouse_to(x=607,y=829)
left_click()
move_mouse_to(x=607,y=749)
left_click()
press_enter()
```



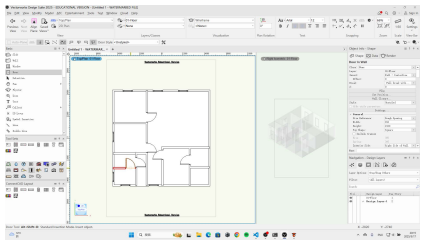
(f)
Insert first window
```
shortcut(combo='shift + d')
move_mouse_to(x=568,y=393)
left_click()
press_enter()
–
–
```



(g)
Insert final window
```
shortcut(combo='shift + d')
move_mouse_to(x=785,y=829)
left_click()
press_enter()
–
–
```

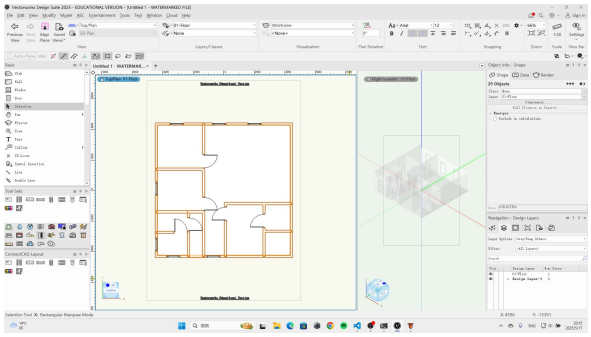


(h)
Insert first door
```
shortcut(combo='alt+shift+d')
move_mouse_to(x=133,y=309)
left_click()
press_enter()
–
–
```



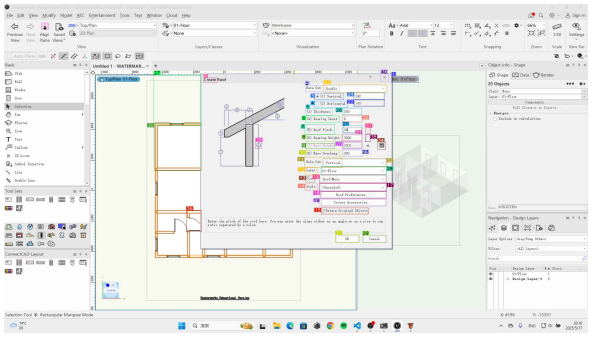
(i)
Insert final door
```
move_mouse_to(x=581,y=749)
left_click()
press_enter()
–
–
–
```

Figure 11. Screenshots of element creations.



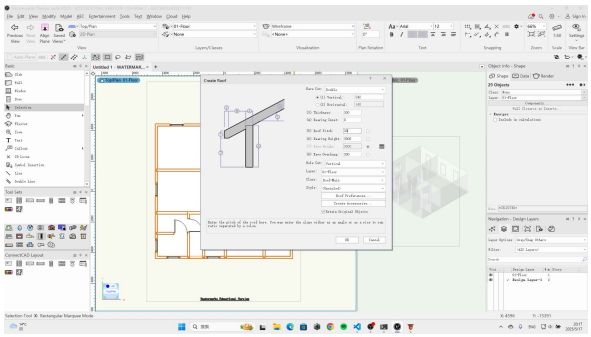
(a)

Select all components
`select_all()`
–
–



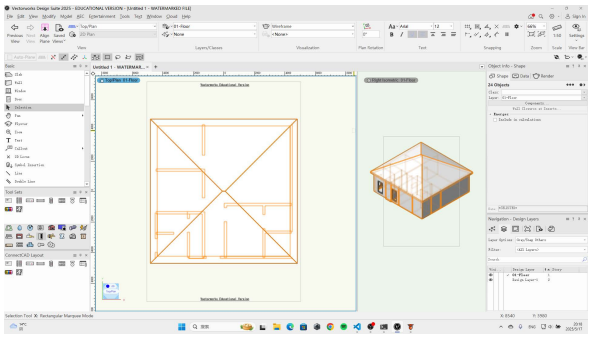
(b)

Active roof tool
`shortcut(combo="ctrl + alt + shift + 1")`
–
–



(c)

Set parameters
`move_mouse_to(1145,353)`
`left_click()`
...
`type_name("30")`



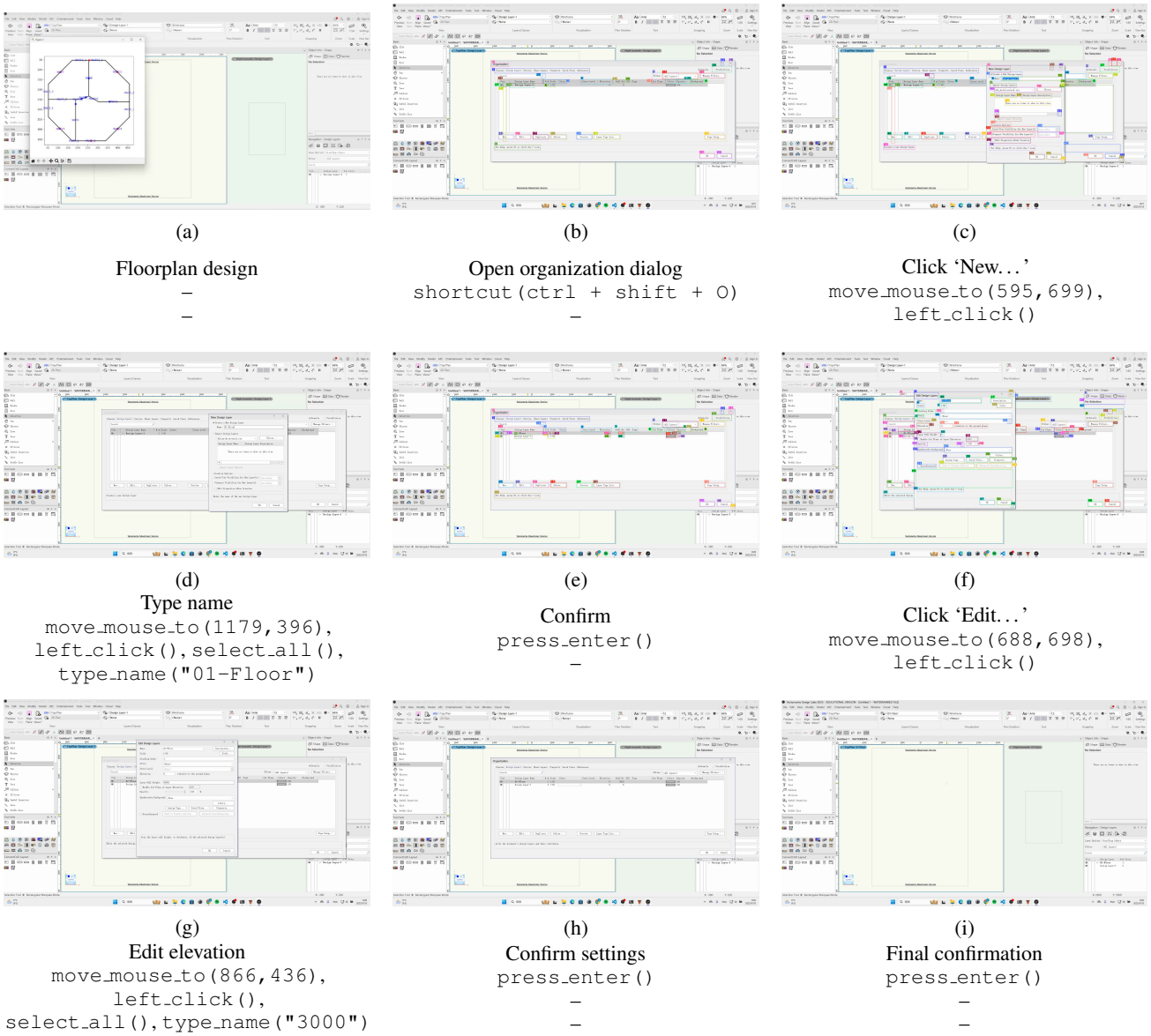
(d)

Confirm roof
`press_enter()`
–
–

Figure 12. Screenshots of roof creation steps.

D.2. Task 16 Action Examples

Given the task: *Generate a building model based on a hand-drawn octagon floorplan, modifying the interior layout to include four rooms instead of three.* The floorplan design and new design layer creation are illustrated in Figure 13, element creation is shown in Figure 14, and roof creation is depicted in Figure 15.



(a)
Floorplan design
–
–

(b)
Open organization dialog
`shortcut(ctrl + shift + O)`
–

(c)
Click 'New…'
`move_mouse_to(595,699),`
`left_click()`

(d)
Type name
`move_mouse_to(1179,396),`
`left_click(),select_all(),`
`type_name("01-Floor")`

(e)
Confirm
`press_enter()`
–

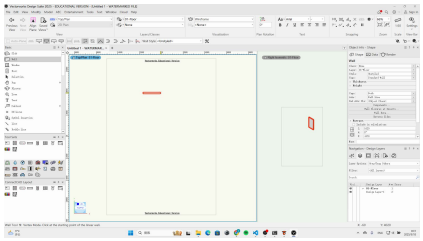
(f)
Click 'Edit…'
`move_mouse_to(688,698),`
`left_click()`

(g)
Edit elevation
`move_mouse_to(866,436),`
`left_click(),`
`select_all(),type_name("3000")`

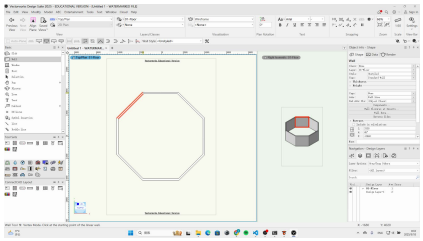
(h)
Confirm settings
`press_enter()`
–
–

(i)
Final confirmation
`press_enter()`
–
–

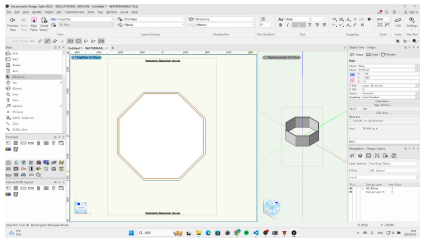
Figure 13. Screenshots of Floorplan Design and Design Layer creation actions.



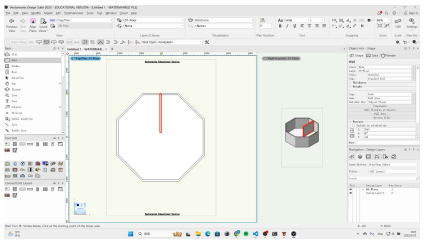
(a)
First external wall
```
shortcut(combo='9')
move_mouse_to(x=641,y=410)
left_click()
move_mouse_to(x=722,y=410)
left_click()
press_enter()
```



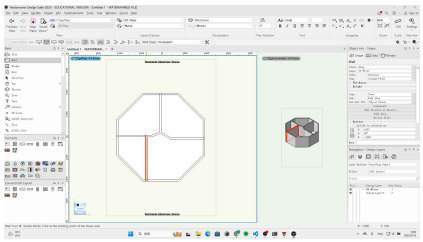
(b)
Last external wall
```
shortcut(combo='9')
move_mouse_to(x=522,y=529)
left_click()
move_mouse_to(x=641,y=410)
left_click()
press_enter()
```



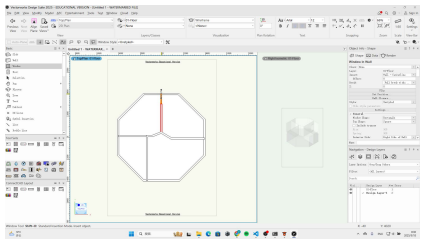
(c)
Create slab by picking external walls
```
shortcut(combo='alt+shift+2')
move_mouse_to(x=681,y=410)
left_click()
...
left_click()
press_enter()
```



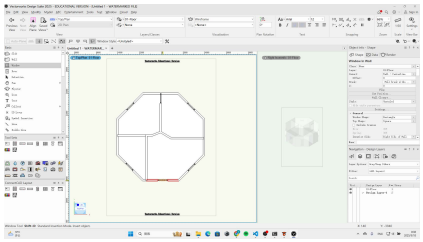
(d)
Create first internal wall
```
shortcut(combo='9')
move_mouse_to(x=722,y=588)
left_click()
move_mouse_to(x=722,y=410)
left_click()
press_enter()
```



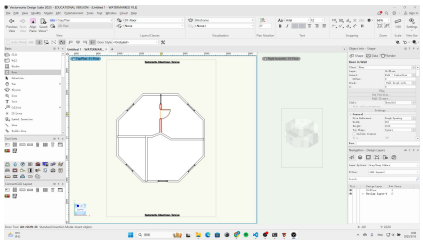
(e)
Final internal wall
```
shortcut(combo='9')
move_mouse_to(x=656,y=810)
left_click()
move_mouse_to(x=656,y=604)
left_click()
press_enter()
```



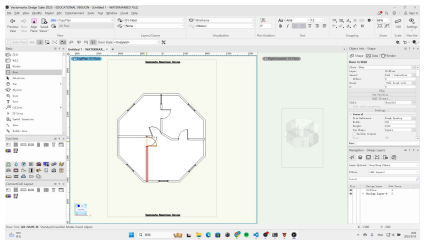
(f)
Insert first window
```
shortcut(combo='shift + d')
move_mouse_to(x=723,y=410)
left_click()
press_enter()
_
_
```



(g)
Insert final window
```
shortcut(combo='shift + d')
move_mouse_to(x=732,y=810)
left_click()
press_enter()
_
_
```

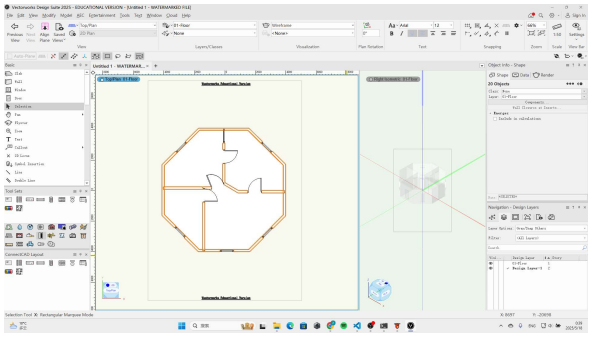


(h)
Insert first door
```
shortcut(combo='alt+shift+d')
move_mouse_to(x=722,y=501)
left_click()
press_enter()
_
_
```



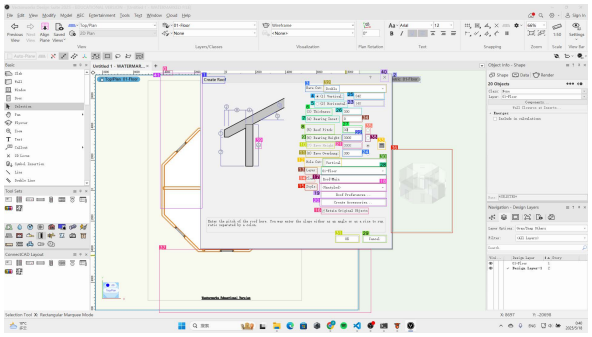
(i)
Insert final door
```
shortcut(combo='alt+shift+d')
move_mouse_to(x=656,y=627)
left_click()
press_enter()
_
_
```

Figure 14. Screenshots of element creations.



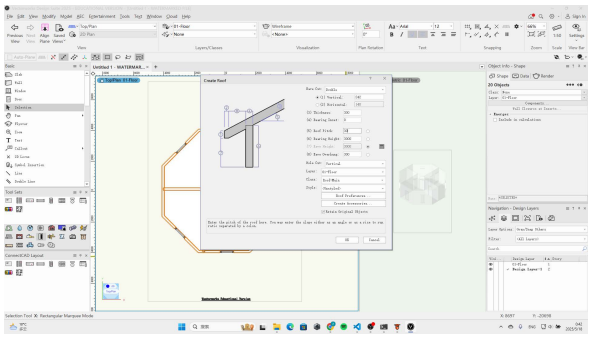
(a)

Select all components
`select_all()`
–
–



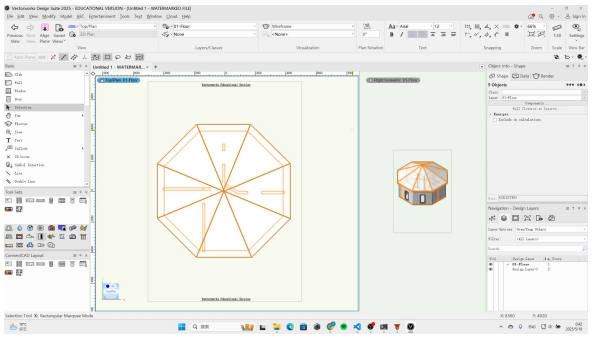
(b)

Active roof tool
`shortcut(combo="ctrl + alt + shift + 1")`
–
–



(c)

Set parameters
`move_mouse_to(1145,353)`
`left_click()`
...
`type_name("30")`



(d)

Confirm roof
`press_enter()`
–
–

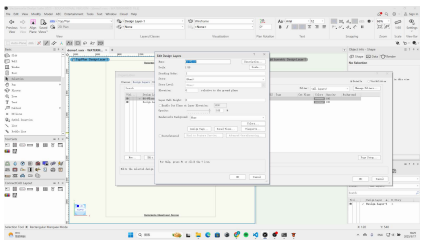
Figure 15. Screenshots of roof creation steps.

D.3. Failure Examples

We present examples of the three types of errors that occurred during the experiment. As illustrated in Figure 16 to Figure 18, these examples demonstrate Grounding Error, Execution Error, and Planning Error, respectively.

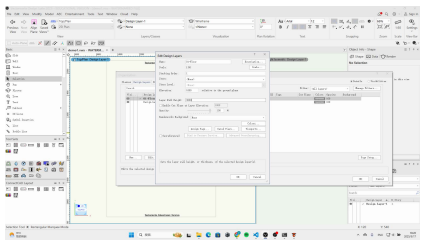
Set the Elevation (Z) to 1000 and the layer wall height (ΔZ) to the default (3000 mm) for the first storey.

(a) Planning info: Elevation (Z) is wrong.



(b)

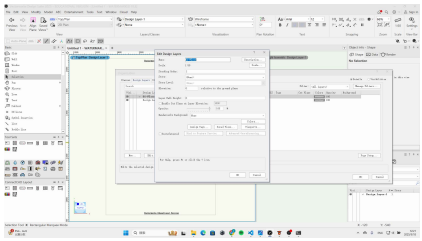
Click 'Edit…'
`move_mouse_to(684,698),`
`left_click()`



(c)

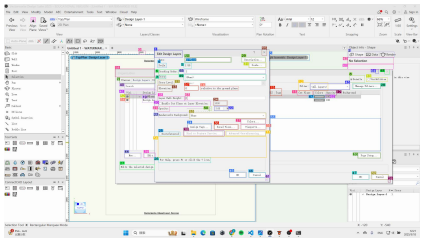
Set elevation
`move_mouse_to(865,383),`
`left_click(),`
`select_all(),type_name(1000)`

Figure 16. A failed task caused by a planning error: the elevation (Z) was incorrectly set to `1000` instead of `0`.



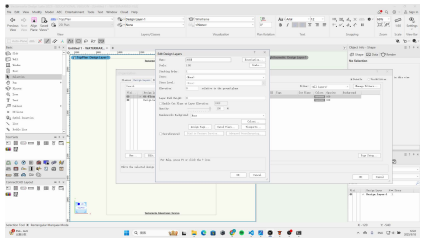
(a)

Click 'Edit…'
`move_mouse_to(684,698),`
`left_click()`



(b)

Grounding error
Detect bounding box of 51 as elevation instead of 63



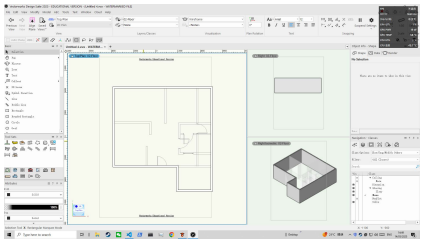
(c)

Edit parameters
`move_mouse_to(950,445),`
`left_click(),`
`select_all(),type_name("3000")`

Figure 17. A failed task caused by a grounding error: the mouse clicked on bounding box 51 instead of 63, resulting in the **Name** being incorrectly changed to `3000`, while the **Elevation** value remained unchanged.

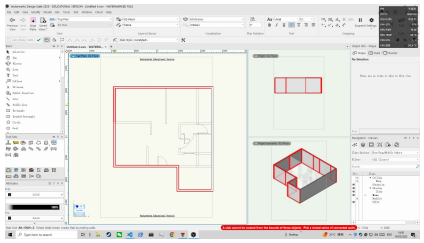
Create the main floor slab for the second floor, covering the area defined by the external walls.

(a) Subtask: Create the slab for the second floor



(b)

Active slab tool
`shortcut(combo = "alt + shift + 2")`



(c)

Create slab
`move_mouse_to(x=585, y=383), left_click(),..., move_mouse_to(x=585, y=383), left_click(),press_enter()`

Figure 18. A failed task caused by an execution error: Create the main floor slab for the second floor, covering the area defined by the external walls. Generated action clicked a same wall twice, which cannot form a closed boundary.

E. Visualization of Generated Models

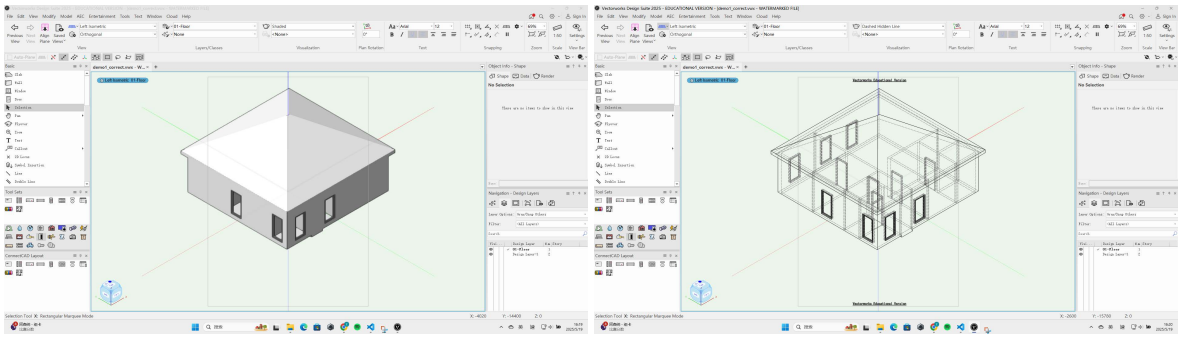


Figure 19. Task 1: *Generate a one-storey office building with a large open workspace occupying most of the floor area. The layout should also include two enclosed meeting rooms, a manager's office, a small pantry, and two restrooms.* Shown are the resulting building model in shaded (left) and wireframe (right) modes.

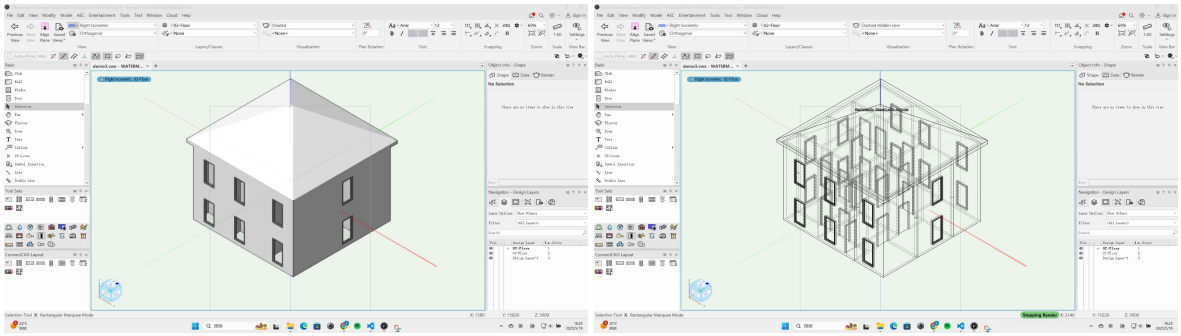


Figure 20. Task 4: *Design a two-storey hospital building with eight distinct rooms. These must include a reception area, two consultation rooms, one minor surgery room, one waiting area, two patient rooms, and one staff room.* Shown are the resulting building model in shaded (left) and Dashed Hidden Line (right) modes.

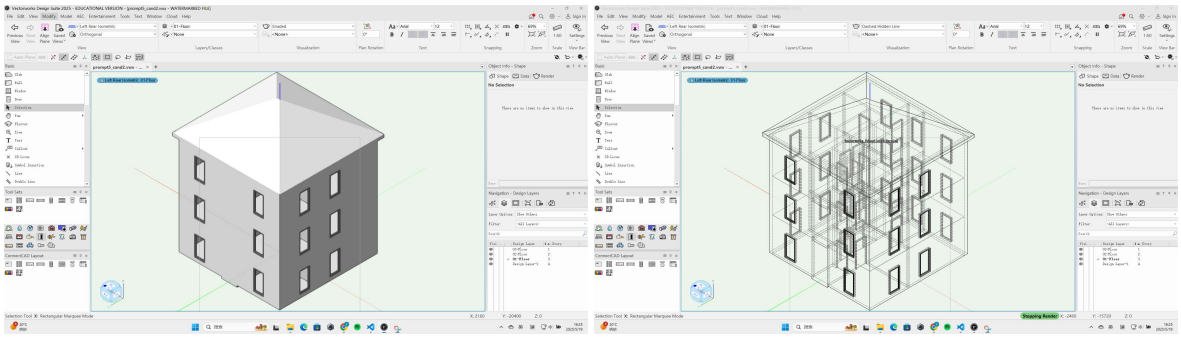


Figure 21. Task 5: *Create a three-storey commercial office building where each floor has the same layout. Each floor must include two large office rooms, a small meeting room, a restroom, and a central corridor. The entrance to the building is located on the ground floor and leads directly to the corridor.* Shown are the resulting building model in shaded (left) and Dashed Hidden Line (right) modes.

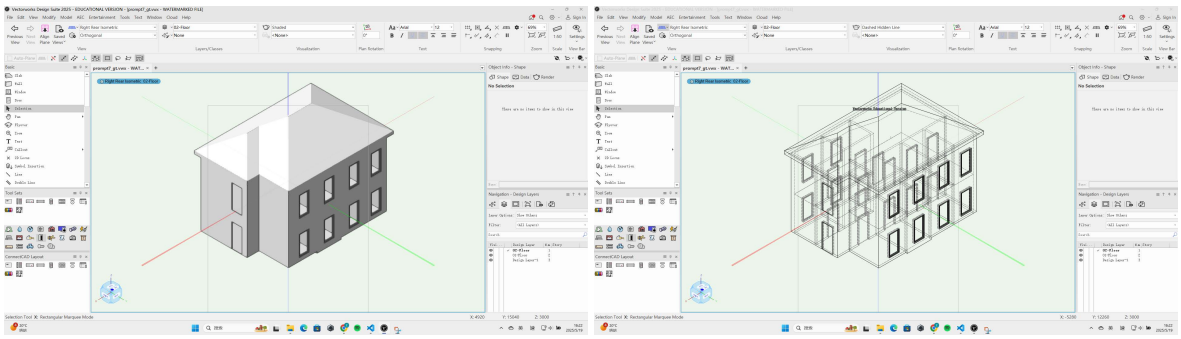


Figure 22. Task 7: *Generate a two-storey building based on the sketch.* Shown are the resulting building model in shaded (left) and Dashed Hidden Line (right) modes.

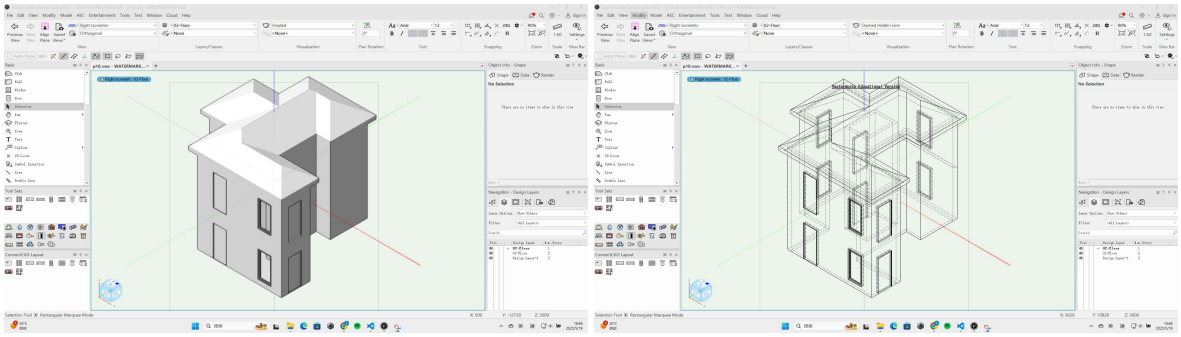


Figure 23. Task 10: *Generate a one-storey building based on the sketch.* Shown are the resulting building model in shaded (left) and Dashed Hidden Line (right) modes.

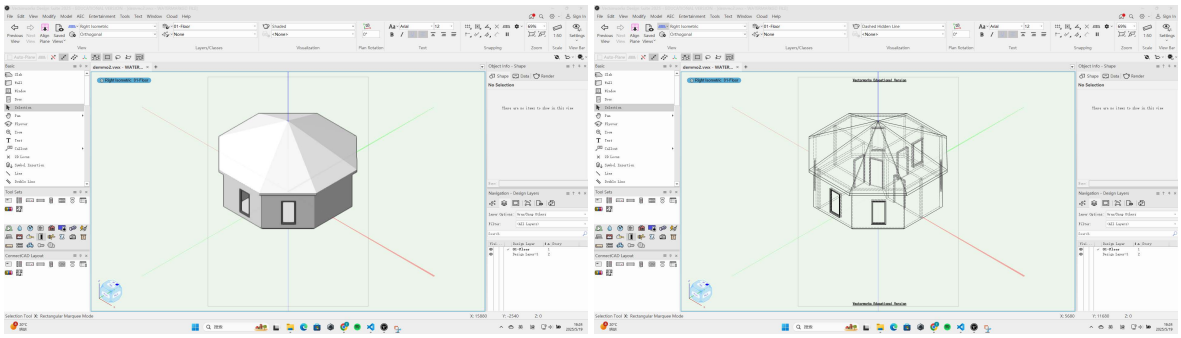


Figure 24. Task 16: *Generate a building model based on a hand-drawn octagon floorplan, modifying the interior layout to include four rooms instead of three.* Shown are the resulting building model in shaded (left) and Dashed Hidden Line (right) modes.

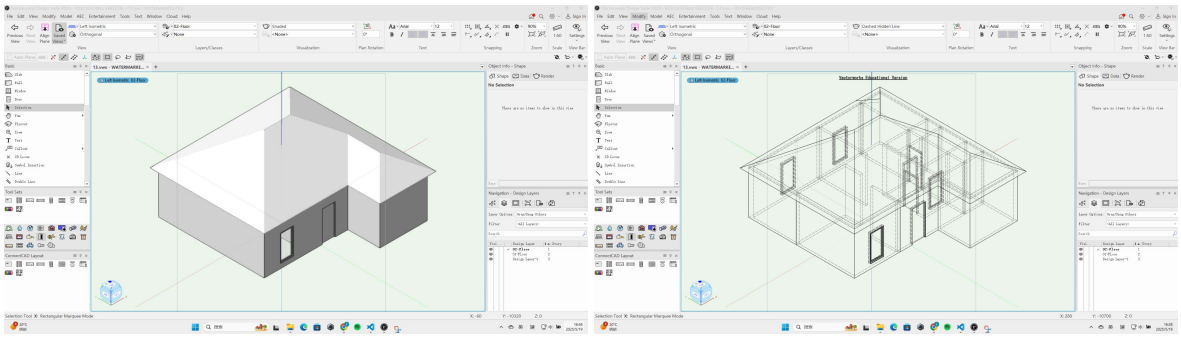
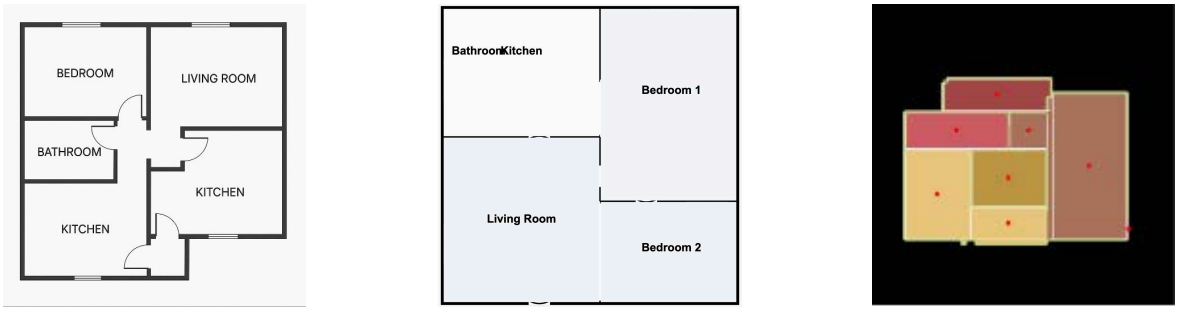


Figure 25. Task 22: *Make a one-floor apartment based on the image but with updates. Add an additional room in the bottom-right corner.* Shown are the resulting building model in shaded (left) and Dashed Hidden Line (right) modes.

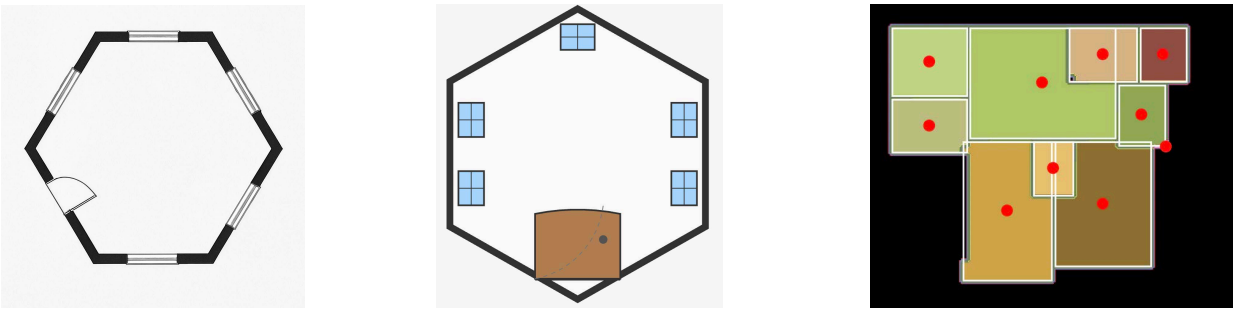
F. Floorplan Generation Comparison

In this section, we present results of floorplan image generation using three different approaches: gpt-image-1, Claude 3.7, and House-GAN++ (Nauata et al., 2021). Figures 26 to 28 show three representative examples. As illustrated, gpt-image-1 produces the most satisfactory results, offering layouts that are not only reasonable but also well-suited for downstream segmentation tasks.



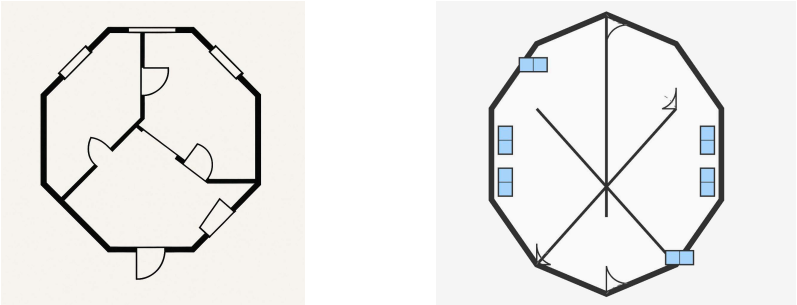
(a) GPT-Image-1 (b) Claude 3.7 (c) House-GAN++

Figure 26. Example 1: *Generate a residential floorplan with 5 rooms: 2 bedrooms, 1 living room, 1 kitchen, and 1 bathroom.*



(a) GPT-Image-1 (b) Claude 3.7 (c) House-GAN++

Figure 27. Example 2: *Design a floorplan with a complex polygonal footprint (hexagonal).*



GAN not supported

(a) GPT-Image-1 (b) Claude 3.7 (c) House-GAN++

Figure 28. Example 3: *Generated floorplans for Task 6 in the Mini Building Benchmark.* The GAN model does not support this task.

G. BIMgent Prompts

G.1. Prompt for High-level Planner

```
You are an assistant acting as a high-level planner for a building design and
    construction project using BIM software. You have been provided with the complete
    floorplan metadata and a design task from the architect. Your role is to outline
    the construction process as a sequence of high-level, logically ordered steps that
    guide the modeling workflow within the BIM environment.

Task Description:
<Task Description>

A structured floorplan including coordinates and types of walls, doors, windows, etc.:
<Interpreted Floorplan Metadata>


Here are some hints to support your decision-making process.
1. Generate high-level construction steps following the typical architectural workflow,
     for example: creating layers, placing walls ...
2. For each construction step, follow these rules:
    - Determine the number of storeys from the task description. For each storey,
        create a corresponding layer.
    - All floors are identical. For each floor, construct all required components based
         on the floorplan.
    - For each component, specify both its name and its assigned floor (e.g., floor2).
...


You must respond strictly in the following format. Do not include any comments,
    explanations, or additional information. Output only the requested data exactly as
    specified below:

{
    "step 1": {
        "class": "layer"
        "component": "layer_floorx",
        "description": "Detailed which design layer should be created currently for the
             current floor.",
    },
    "step 2": {
        "class": "external walls"
        "component": "wallx_floorx, ...",
        "description": "Detailed description of what needs to be done"
    },
    "step x": {
        "class": "the class of the drawing components."
        "component": "the components in specific floor that should be draw here",
        "description": "Detailed description of what needs to be done"
    }
}
```

Listing 1. Example prompt used for High-level Planner

G.2. Prompt for Low-level Planner

```
You are an assistant acting as a low-level planner, responsible for generating detailed
     action steps based on software guidance and the provided high-level plan.

your current task:
<Current General Step>

tool guidance:
<Software Documentation Retrieved via RAG>

floorplan_metadata:
<Interpreted Floorplan Metadata>


Here are some hints to support your decision-making process.
You have to decide two types of actions:
Vision-Driven: Actions for which explicit coordinates are not yet provided, including
    shortcut-based operations.
...

Pure-Action: Actions for which coordinates are explicitly provided in the floorplan
    metadata, such as wall creation.
<Action Definitions>
<Speculative Multi-action Execution>
...

You must respond strictly in the following format. Do not include any comments,
    explanations, or additional information. Output only the requested data exactly as
    specified below:

{{
    "sub_step_1": {{
        "action name": "...",
        "action_type": "Vision-Driven"
        "description": "Detailed current step's goal"
    }},
    "sub_step_2": {{
        "action name": "...",
        "action_type": "Pure-Action"
        "actions": ['action1()', 'action2(x=..., y=...)', ...],
        "coordinates": [[x1, y1], ...],
        "description": "Detailed current step's goal"
    }},
    "sub_step_x": {{
        "action name": "...",
        "action_type": "Pure-Action"
        "actions": ['action1()', 'action2(x=..., y=...)', ...],
        "coordinates": [[x1, y1], ...],
        "description": "Detailed current step's goal"
    }}
    "sub_step_x": {{
        "action name": "...",
        "action_type": "Vision-Driven"
        "description": "Detailed x step's goal"
    }}
}}
```

Listing 2. Example prompt used for Low-level Planner

G.3. Prompt for Action Generator

```
You are an action generator. Your task is to implement the provided actions by
    combining and sequencing them effectively to accomplish the given subtask. Ensure
    that all actions are context-aware, precise, and optimized for the tools and
    workflows in Vectorworks 2025. Below is some helpful information to assist your
    decision-making.

you will be provided with an image of the current screenshot image, which is already
    segmented, and the meta information of the provided image.

Your task:
<Current Substep>

meta_information of the labeled image:
<Interpreted Floorplan Metadata>

Feedback from Supervisor:
<Reasons>

Based on the image and metadata, you must respond by following the rules below:
1. Generate a workflow consisting of all necessary actions required to complete the
    given task.
2.Your output for the actions field must be a plain string representing a list of
    actions in the following format: "['action1()', 'action2(x=..., y=...)', ...]". Do
    not include any additional text, explanation, or formatting.
...

You must respond strictly in the following format. Do not include any comments,
explanations, or additional information. Output only the requested data exactly as
    specified below:
{{
    "action name": "...",
    "actions": "['move_mouse_to(x=, y=)', ... ]"
}}
```

Listing 3. Example prompt used for Action Generator

G.4. Prompt for Supervisor – Vision-Driven Workflow

```
You will be provided with a screenshot of the current GUI state.

Additionally, you are given the current task content:
<Current Substep>

The list of executed actions:
<Executed Actions>


Based on the information provided, you must respond according to the following rules:
approved_value:
1. Open Dialog: If the task is to open a dialog, check whether the dialog is visible in
     the screenshot.
2. Enter Name: If the task is to enter a name, verify that the correct name has been
    typed into the appropriate input field.
...

reasons:
1. If the result of approved_value is "fail", you must provide clear reasoning for your
     decision. If the result is "success", simply return "success" in this field.
...

You should respond strictly in the following format, and you must not output any
    comments, explanations, or additional information. Don't include anything beside
    the requested data represented in the following format

approved_value:
success/fail

reasons:
...
```

Listing 4. Example prompt used for Supervisor – Vision-Driven Workflow

G.5. Prompt for Supervisor – Pure-Action Workflow

```
You will be provided with an image that contains the object information of the created
    elements and the executed actions.

Additionally, you are given the current task content:
<Current Substep>

The list of executed actions:
<Executed Actions>


You must respond by following the rules below:
approved_value:
At the right side of the image, information about the most recently created or selected
     object is displayed. You must identify the component name and determine whether it
     corresponds to the current task content. If the created object matches the task,
    respond with "success"; otherwise, respond with "fail".

actions:
If the object does not correspond to the task, you must regenerate a corrected list of
    actions based on the current executed_actions, following the defined action
    generation rules:
...


You should respond strictly in the following format, and you must not output any
    comments, explanations, or additional information. Don't include anything beside
    the requested data represented in the following format

approved_value:
success/fail

actions:
{{
    "action name": "...",
    "actions": "['move_mouse_to(x=, y=)', ... ]"
}}
```

Listing 5. Example prompt used for Supervisor – Pure-Action Workflow