

MCP4IFC: IFC-BASED BUILDING DESIGN USING LARGE LANGUAGE MODELS

Bharathi Kannan Nithyanantham¹ **Tobias Sesterhenn²** **Ashwin Nedungadi¹**

Sergio Peral Garijo² **Janis Zenkner²** **Christian Bartelt²** **Stefan Lüdtke¹**

¹University of Rostock ²Technical University of Clausthal

bharathikannan.nithyanantham@uni-rostock.de

tobias.sesterhenn@tu-clausthal.de

ABSTRACT

Bringing generative AI into the architecture, engineering and construction (AEC) field requires systems that can translate natural language instructions into actions on standardized data models. We present MCP4IFC, a comprehensive open-source framework that enables large language models (LLMs) to directly manipulate Industry Foundation Classes (IFC) data through the Model Context Protocol (MCP). The framework provides a set of BIM tools, including scene querying tools for information retrieval, predefined functions for creating and modifying common building elements, and a dynamic code-generation system that combines in-context learning with retrieval-augmented generation (RAG) to handle tasks beyond the predefined toolset. Experiments demonstrate that an LLM using our framework can successfully perform complex tasks, from building a simple house to querying and editing existing IFC data. Our framework is released as open-source to encourage research in LLM-driven BIM design and provide a foundation for AI-assisted modeling workflows. Our code is available at <https://github.com>Show2Instruct/ifc-bonsai-mcp>.

1 INTRODUCTION

Building Information Modeling (BIM) has transformed how the architecture, engineering and construction (AEC) fields work. Instead of using separate 2D drawings, teams share a single 3D model that contains all the important details about a building (Succar, 2009). This enables seamless collaboration and more efficient coordination throughout the project lifecycle. These models can include not only building meshes, but also information about materials, costs, schedules, and more. The Industry Foundation Classes (IFC) standard defines a common schema for storing and sharing BIM data, so different software platforms can use the same models (ISO 16739-1:2024, 2024). However, creating, editing, and maintaining BIM models remains complex and is typically handled by experts. Meanwhile, rapid progress in Large Language Models (LLMs) suggests a new paradigm for interacting with complex software. Modern LLMs can interpret natural language instructions and act as intelligent agents that plan and execute tool calls to achieve user goals (Wang et al., 2023; Yao et al., 2023). This presents an exciting possibility in the BIM context: an AI assistant that enables architects and engineers to describe desired changes or questions in natural language and have the model automatically execute corresponding BIM operations. Recent studies have begun exploring natural language interfaces for BIM, focusing on either model generation and editing (e.g., Du et al. (2024a); Deng et al. (2025)) or on information querying (e.g., Hellin et al. (2025); Li et al. (2025a)). However, most existing systems rely on high-level APIs from proprietary software such as Revit (Autodesk Revit, 2025) or Vectorworks (Vectorworks, 2025). These APIs expose operations at the application layer with their own schema and do not interact directly with the IFC data structure.

To address this limitation, we present MCP4IFC, a framework that leverages the Model Context Protocol (MCP) (Model Context Protocol, 2025) to enable LLMs to read, create, and edit IFC models directly via high-level tool calls. Our framework operates as a local MCP server and utilizes the open-source IfcOpenShell library (IfcOpenShell, 2025) together with the Bonsai add-on (Bonsai, 2025) in Blender. The MCP is an open standard that provides a single protocol that any LLM client can use (Hou et al., 2025). Some projects have already explored connecting MCP with 3D modeling

environments. For instance, the Blender MCP integration (Ahujasid, 2025) generates Blender API code through a code execution tool, and a Bonsai MCP plugin (JotaDeRodriguez, 2025) exposing a few BIM operations to query IFC models and place objects. Our framework introduces a complete and standardized solution that uses IfcOpenShell as the main toolkit for working with IFC data and supports full query, create, and edit operations. All functionalities are implemented via IfcOpenShell API calls, with Bonsai acting only as a wrapper to interact with Blender. As all components are open source, we are not constrained by closed APIs or automation limits of commercial BIM software (Autodesk Revit, 2025; Vectorworks, 2025; Graphisoft, 2025). This modular design makes it straightforward to adapt the tools to proprietary environments.

A key innovation of our framework is combining In-Context Learning (ICL) (Brown et al., 2020) with Retrieval-Augmented Generation (RAG) (Lewis et al., 2020) to dynamically generate Python code for IFC-specific tasks. ICL refers to the model’s ability to learn new tasks or patterns during inference by simply showing examples in the prompt. ICL is the core of our framework, which guides the model to generate a specific tool by providing detailed steps. This allows the model to go beyond the provided fixed toolset. When the model’s prior knowledge is insufficient (e.g., with the IfcOpenShell API), the LLM can invoke an optional RAG tool. This module retrieves relevant documentation and code examples from a local vector store and injects them into the prompt. The retrieved information provides additional context for ICL to generate accurate code. This design follows the retrieval-augmented code generation approach, where LLMs search large codebases or external documentation to produce accurate and context-aware code (Zhou et al., 2022; Yang et al., 2025b; Wang et al., 2024b).

We demonstrate that LLMs can design and modify IFC models by following step-by-step natural-language instructions. For example, an LLM can incrementally construct a small house using only standardized tool calls and achieve scene-query performance competitive with state-of-the-art methods (Hellin et al., 2025). These indicate that standardized tool use enables LLMs to work effectively across the BIM lifecycle. To our knowledge, our framework is the first comprehensive MCP server that allows LLMs to create, edit, and query IFC models while also supporting dynamic tool generation.

Our main contributions are:

- 1. An open-source MCP framework for IFC:** Our framework offers a comprehensive set of more than 50 BIM tools, enabling LLMs to create, modify, and query IFC models through standardized MCP tool calls.
- 2. Dynamic tool expansion through ICL and RAG:** Our framework combines ICL with a RAG system to support on-the-fly code generation for IFC-specific tasks beyond a pre-defined toolset.
- 3. Demonstrated end-to-end BIM manipulation:** We show that existing multimodal LLMs can successfully perform IFC querying, editing, and generation tasks using only natural language and tool calls, without direct user interaction with BIM software.

2 RELATED WORK

Applying LLMs in the AEC field is an active, rapidly growing area of research. Our work focuses on tool-augmented LLMs, where models can invoke external tools to create and modify IFC-based BIM data. This section reviews related studies on LLMs for tool use, their application to BIM and 3D design workflows, and their ability to understand spatial information.

2.1 LLMs AS TOOL-USING AGENTS

Augmenting LLMs with external tools has become an active area of research in recent years. This line of work focuses on enabling models to take actions in the external world via tool calls. ReAct (Yao et al., 2023) introduced a framework where an LLM iteratively plans actions, calls external tools, observes the results, and then uses that feedback to decide the next step. Toolformer (Schick et al., 2023) showed that LLMs can learn when and how to use APIs without requiring task-specific labels. Approaches such as Gorilla (Patil et al., 2023) train LLMs on large, diverse API specifications and use retrieval to select the appropriate function. Datasets like ToolBench (Qin et al., 2023)

provide thousands of tool-use instructions for model training. As tool-use capabilities become more common, major LLM providers expose APIs that allow models to call external functions or tools directly. However, these interfaces are not compatible across providers, leading to integration and scalability challenges. The Model Context Protocol (MCP) (Model Context Protocol, 2025) addresses this issue by defining a standardized interface for tool discovery and invocation of these tools over HTTP or local IPC. MCP consists of two main components: an AI Client (typically the LLM application) and an MCP Server, which exposes tools, along with their descriptions and input/output formats using JSON Schema. Because the protocol is standardized, any AI Client can connect to the MCP Server without requiring custom integration code. In our work, we adopt MCP for IFC-based BIM tasks.

2.2 LLMS FOR BIM AND 3D DESIGN

Recent work has explored how LLMs can assist in the design of buildings and 3D structures. This refers to tasks such as creating and modifying elements, and retrieving information from models for question answering or more advanced tasks, such as clash detection. Most systems are designed to work on either generation/editing (Deng et al., 2025; Jang et al., 2024; Duggempudi et al., 2025; Du et al., 2024b; Fernandes et al., 2024; Yuan et al., 2024) or information retrieval (Zheng & Fischer, 2023; Hellin et al., 2025; Li et al., 2025a; Guo et al., 2025; Liu & Chen, 2025). We refer to Table 1 for a categorical overview.

BIM Creation and Modification Several systems let LLMs generate code that interacts directly with proprietary BIM software APIs. Text2BIM (Du et al., 2024a) and previous work (Du et al., 2024b) employ multiple LLM agents to call custom tools built on the Vectorworks API to construct BIM models from textual descriptions. NADIA (Jang et al., 2024) and Text-to-Layout (Duggempudi et al., 2025) translate user instructions into intermediate JSON actions (e.g., create, modify, retrieve, delete) that are then converted into executable Revit API calls. Dong et al. (2025) uses the AutoGen framework (Wu et al., 2024) with the Revit API to coordinate specialized agents for wall generation, clash detection, and information retrieval, while Fernandes et al. (2024) takes a simpler approach by representing BIM data as CSV files and exposing a fixed set of operations (e.g., hiding or coloring elements). Although these systems enable fine-grained model manipulation, they remain tightly coupled to closed-source platforms, limiting portability and extensibility. A different direction extends beyond APIs by simulating user interactions through the software’s graphical interface. BIMgent (Deng et al., 2025) enables an LLM to control BIM software via mouse clicks and keystrokes, using screenshots of the current model state as visual feedback for iterative refinement. Similarly, 3D-PreMise (Yuan et al., 2024) integrates rendered images of generated 3D shapes to help the model construct and improve them within Blender. Inspired by these works, our system also incorporates tools for visual feedback, allowing the LLM to assess and respond to the current state of model generation.

BIM Scene Querying Beyond generative workflows, LLMs have also been employed for retrieving information from existing BIM models. For instance, Zheng & Fischer (2023) builds a framework that preprocesses data into a MongoDB database, enabling an LLM to retrieve information by iteratively translating natural-language prompts into database queries. Hellin et al. (2025) performs agentic information retrieval, using a fixed library of functions to retrieve data directly from an IFC file. Li et al. (2025a) integrates a tree-based spatial indexing method with a multi-agent LLM framework, enabling the decomposition of spatial queries into geometric reasoning subtasks for efficient retrieval of 3D spatial relationships. Guo et al. (2025) presents ARCBIM, which uses multiple specialized LLMs and an alignment-refinement coder to convert natural language into executable code for BIM databases. Similarly, BIMCoder (Liu & Chen, 2025) translates natural language queries into SQL-like or IFCQL statements.

The main limitation of existing generative systems is their tight coupling to a specific platform, such as Revit (Jang et al., 2024; Duggempudi et al., 2025; Dong et al., 2025; Fernandes et al., 2024) or Vectorworks (Du et al., 2024a;b). This approach lacks portability and requires significant development to support other platforms. Our MCP4IFC framework addresses this challenge by building directly on the Industry Foundation Classes (IFC) standard (ISO 16739-1:2024, 2024). By operating on this neutral, open-standard data representation, our server can be easily extended to

Paper	Gen.	Edit.	Retr.	Framework	Input	Output	Platform
Du et al. (2024a)	✓	✓	(✓)*	Multi-agent	Text	API calls (Python DSL)	Vectorworks
Deng et al. (2025)	✓	✓	✗	Multi-agent	Text + Image	GUI DSL	Tool-agnostic
Jang et al. (2024)	(✓)*	✓	✗	Single-agent	Text	JSON → Revit API	Revit
Duggempudi et al. (2025)	✓	✗	✗	Single-agent	Text + JSON	API calls (via intermediate JSON)	Revit
Du et al. (2024b)	✓	✓	✗	Single-agent + RAG	Text	API calls (Python DSL)	Vectorworks
Dong et al. (2025)	✓	✓	✓	Multi-agent	Text	API calls	Revit
Fernandes et al. (2024)	✗	✓	✗	Single-agent	Text + CSV	API calls (Python DSL)	Revit
Yuan et al. (2024)	✓	✓	✗	Single-agent	Text + Image	Python Code	Blender
Zheng & Fischer (2023)	✗	✗	✓	Single-agent	Text	MongoDB query	Tool-agnostic
Hellin et al. (2025)	✗	✗	✓	Single-agent	Text	IfcOpenShell calls (Python DSL)	Tool-agnostic
Li et al. (2025a)	✗	✗	✓	Multi-agent	Text	Spatial Reasoning Calls (Python DSL)	Tool-agnostic
Guo et al. (2025)	✗	✗	✓	Multi-agent	Text	DSL code	Tool-agnostic
Liu & Chen (2025)	✗	✗	✓	Single-agent	Text	IFCQL query code	Tool-agnostic
MCP4IFC (ours)	✓	✓	✓	Single-agent	Text + Image	Tool calls (IfcOpenShell)	mostly Tool-agnostic

Table 1: **Comparison of LLM-based frameworks for BIM and 3D design tasks.** Checkboxes (✓, ✗) indicate whether the system supports BIM generation (Gen.), editing (Edit.), or information retrieval (Retr.). * indicates partial or limited support.

other platforms that support IFC. This allows a single set of tools to be developed once and used anywhere in the BIM ecosystem.

2.3 LLMs FOR SPATIAL UNDERSTANDING

Working with BIM data requires models to understand complex 3D spatial structures, including geometry, topology, and the relationships between building components. To address this gap, research focuses on vision-language grounding, which connects textual commands to multi-modal 3D data Li et al. (2025b); Yang et al. (2025a); Jia et al. (2024); Zhang et al. (2025b). Researchers are exploring various strategies to achieve this. For instance, some utilize structured representations such as 3D scene graphs to capture semantic and relational data Zhang et al. (2025a); Engelbracht et al. (2024); Chen et al. (2022); Longo et al. (2025); Schulz et al. (2023). Other works explore implicit neural representations, such as NeRFs Mildenhall et al. (2020) to model geometry as a continuous function, or develop hybrid methods that integrate spatial properties directly into the LLM’s framework using spatial tokens or 3D-informed architectures Mao et al. (2025); Ma et al. (2025); Hong et al. (2023). Our framework provides screenshot tools that give visual context to the LLM and enable spatial reasoning in BIM environments. This allows the model to inspect and verify its outputs using its current image-understanding capabilities. While this does not fully solve spatial reasoning, it is a practical step that improves the reliability of BIM interactions and lays the foundation for stronger spatial models in the future.

3 METHODOLOGY

The MCP4IFC framework connects LLMs with IFC-based BIM workflows via the Model Context Protocol. The framework offers a structured interface that allows LLMs to query, create, and edit IFC elements in a standardized way. This section explains the overall system architecture and the toolset that allows LLMs to interact directly with IFC data.

3.1 SYSTEM ARCHITECTURE AND WORKFLOW

The MCP4IFC framework consists of three main components, as shown in Figure 1: the AI Client, the MCP Server, and the Blender add-on. The AI Client is an LLM interface that interacts with the user and displays the output. The MCP Server is a Python program that implements the MCP standard and manages all communication between the AI Client and the Blender add-on. It exposes a structured list of BIM tools, each defined as a JSON object that specifies its name, description, and input/output schemas. These tool definitions act as high-level wrappers, while the actual BIM operations are executed inside the Blender add-on. The add-on uses the IfcOpenShell library to manipulate IFC data and the Bonsai extension to update geometry in Blender’s 3D scene. Two communication links connect these components: the AI Client sends JSON-RPC messages to the MCP Server, and the server forwards commands to the Blender add-on through a socket connection.

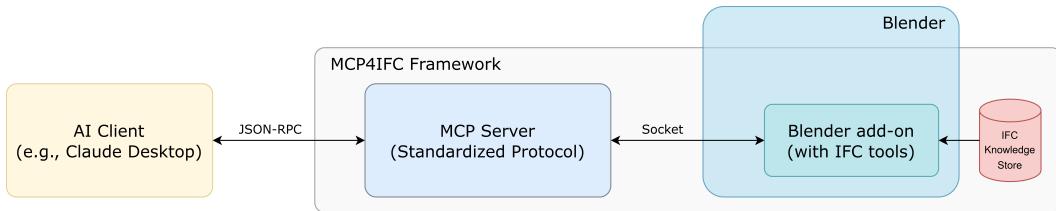


Figure 1: Overall system architecture of our MCP4IFC framework

A detailed workflow is shown in Figure 2. When the AI Client connects to the MCP Server, it requests the available tools. After receiving the tool list in JSON format, the client adds it to the system prompt, so the model can understand the available functions through in-context learning. When a user provides a prompt, such as *add a 5-meter wall*, the LLM selects an appropriate tool, generates structured parameters, and sends a JSON-RPC request to the MCP server. The server validates this request and forwards it via the socket connection to the Blender add-on. The add-on

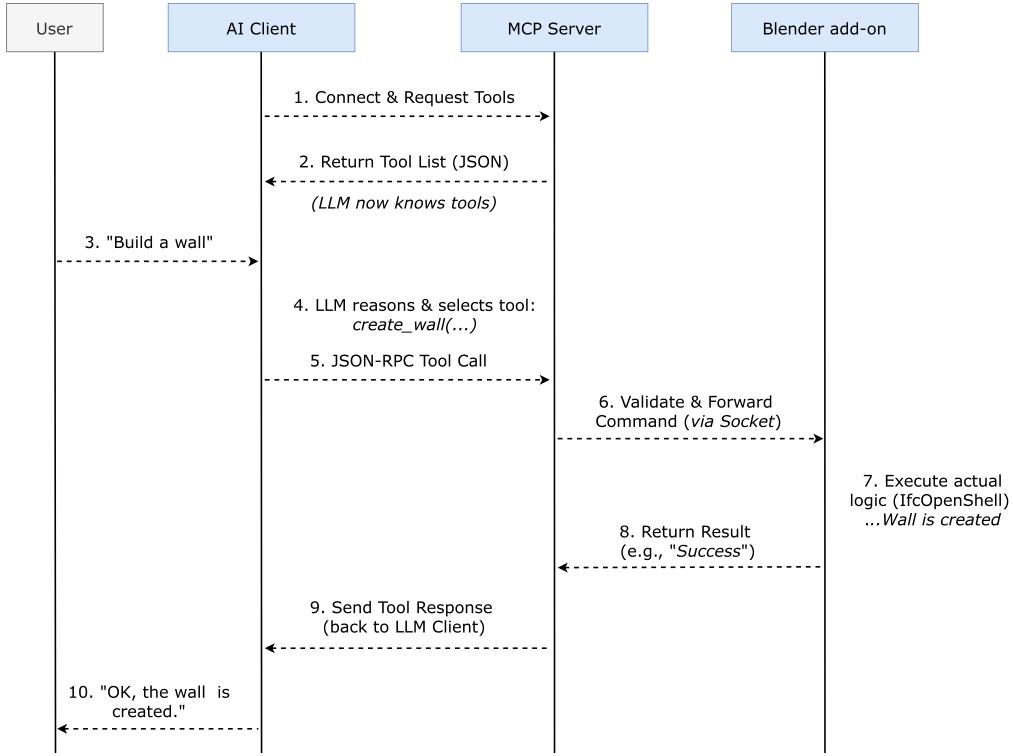


Figure 2: Workflow of a simple wall creation request through the MCP4IFC framework.

executes the action and returns a structured result to the server, such as a success message or an error. The LLM analyzes it, determines the next step, and continues the loop until the task is completed.

3.2 TOOLSET IMPLEMENTATION

The MCP server includes a comprehensive list of tools. Each tool is defined with a clear name, description, input parameters, and expected outputs. The following sections describe the main tool categories and how they are implemented within the MCP4IFC framework.

3.2.1 SCENE QUERY AND CONTEXT TOOLS

Before an LLM can modify a BIM model, it needs to understand the existing elements in the scene. MCP4IFC provides scene and data querying tools for this purpose. These tools return a compact, structured summary of the model instead of exposing the entire IFC file. The summary lists each top-level element, including important fields such as the element's GUID, IFC type, and basic properties, such as name, type, and location. This gives the LLM a high-level view of the scene that is easy to parse and reason about. Using this information, the model can answer questions like *How many walls exist and what are their lengths?* or select the correct object to change based on a given identifier. This design offers several advantages over directly exposing IFC or related formats such as ifcXML or ifcJSON. Raw IFC data is long, highly detailed, and not efficient for LLMs to process. In contrast, the structured JSON summary retains only the information needed for LLM reasoning.

In addition to structured data, MCP4IFC supports multimodal inputs to provide spatial context. These tools capture screenshots of the modeling environment, either the full application window or only the 3D view, to show the model's current state. This helps the model verify changes visually, similar to how human designers work by observing the scene and refining results iteratively.

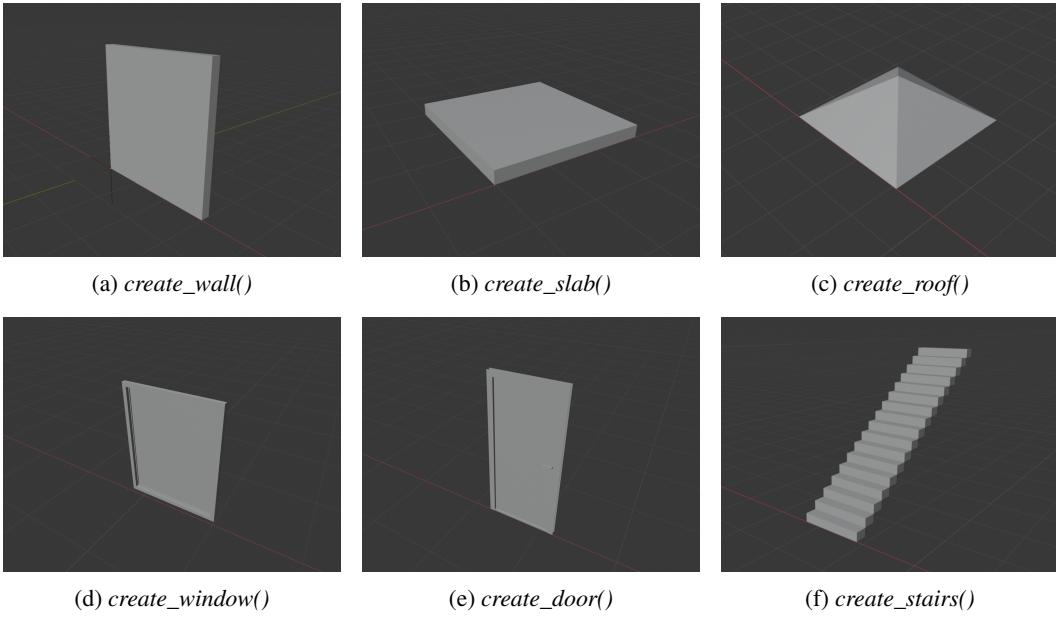


Figure 3: Example IFC building elements generated through predefined tool functions. These functions also support a variety of parametric input arguments.

3.2.2 PREDEFINED IFC ELEMENT CREATION TOOLS

The LLM can use predefined tools to create, modify, or delete elements in the IFC model. These tools perform standard IFC operations on core building components, including walls, slabs, roofs, doors, windows, and stairs, as demonstrated in Figure 3. Each tool has a set of well-defined input parameters that specify the properties and placement of the element. For example, the wall creation tool accepts a start and end point along with height and thickness values to generate an IFC wall element. Another variant of this tool takes a list of points and makes a chain of walls to outline a room. Slabs are created from closed polygon outlines and a thickness parameter, while roofs can be generated from a 2D outline or by referencing supporting walls. Doors and windows are inserted into existing walls by providing the wall GUID along with size and position. Every newly created element will have a unique IFC GUID, which is returned to the AI Client for continued reference in the next operations. In addition, the framework provides editing and deletion tools that enable an iterative workflow in which the model builds, checks, and refines the design step by step. The quality of the final design depends on how effectively the LLM selects the appropriate tools and configures their parameters. In practice, the LLM usually begins by creating a basic layout, then adds and refines elements using the predefined tools until the desired configuration is achieved.

3.3 DYNAMIC CODE GENERATION TOOLS

Predefined tools cover most of the common BIM operations, but they cannot represent every possible task. Real building models often require custom shapes, specific IFC manipulations, or unique design logic that goes beyond the fixed toolset. To support these cases, MCP4IFC includes dynamic code generation tools, which allow the LLM to write and execute Python scripts during inference. Each script runs inside a secure Blender environment and returns structured feedback to the AI Client. This enables handling new element types, advanced geometry generation, and specialized workflows without requiring manual development of new predefined tools.

Dynamic tool usage falls into two main categories: IFC code generation and mesh generation. For operations involving IFC data that are not covered by existing tools, the LLM can directly generate Python code using the IfcOpenShell API. Since these libraries are publicly available online, LLMs have typically seen them during pretraining. This enables them to generate valid Python scripts that can manipulate IFC data directly. The code is executed in a sandboxed environment that blocks system access and restricts modules to approved libraries. After execution, the system returns logs,

error messages, and object identifiers, which the LLM can analyze to refine and verify its operations. To further improve reliability, the MCP Server integrates a retrieval-augmented generation (RAG) system. It stores IfcOpenShell documentation, IFC schema definitions, and example scripts in a local vector store database. When needed, the LLM can retrieve relevant context using semantic search tools. This ensures the generated code uses the correct and up-to-date APIs.

Dynamic code generation also extends to direct geometry creation. Predefined IFC creation tools cannot easily represent complex elements such as curved ramps, bridges, or freeform structures. In such cases, the framework allows the LLM to generate Python code using external 3D geometry libraries. In principle, any compatible Python library can be used, as long as the LLM has sufficient knowledge of it from pretraining to generate correct API calls. The framework uses trimesh (Dawson-Haggerty et al., 2019) for its simplicity, but other libraries can also be integrated (Khan et al., 2024). The LLM determines the appropriate IFC class and a few semantic properties for the generated mesh, though it may not fully generate the IFC Space or the relationships between objects. This could be future work. This enables open-ended design where the LLM can create complex geometry procedurally. Future versions of the framework can incorporate more advanced parametric modeling libraries to further enhance complex and advanced design.

4 EXPERIMENTS AND RESULTS

The following experiments aim to demonstrate the feasibility and practical functioning of the MCP4IFC framework. Rather than providing an exhaustive quantitative benchmark, our goal is to illustrate that the proposed system can perform key BIM operations, i.e., querying, editing, and generation, through standardized tool use and dynamic code generation. The experiments serve as a proof of concept to validate the end-to-end workflow and to highlight potential research directions.

4.1 SCENE QUERYING

This section evaluates the performance of the MCP4IFC system on scene querying tasks. For this, we use a subset of the IFC-bench-v1 dataset (Hellin et al., 2025) of 65 questions associated with the *Duplex* project’s architectural (ARC) and mechanical, electrical and plumbing (MEP) models. A visualization of the Duplex house can be seen in Figure 4. Following Hellin et al. (2025), the dataset can be divided into three categories of questions:

- **Direct (27 tasks):** Queries solvable by directly extracting information from the IFC model (e.g., retrieving a specific object’s GUID).
- **Indirect (16 tasks):** Queries requiring aggregation or computation (e.g., counting all radiators or calculating a total area).
- **Insufficient (22 tasks):** Queries where the necessary information is not present in the ifc model, potentially requiring the LLM to make assumptions or use its own expertise to solve the question (e.g., estimating how much of the materials can be recycled at the end-of-life).

We conduct the experiment on two LLMs, Claude Sonnet 4.5 and GPT-5 mini, and manually check answers for correctness, i.e. without using the LLM-as-Judge evaluation framework as conducted by Hellin et al. (2025). As shown in Table 2, GPT-5 mini works best achieving an accuracy of 83.1%, compared to 75.4% accuracy of Sonnet 4.5. Hereby the benefit of the gpt model primarily stems from a better performance on *direct* retrieval questions, solving 26 tasks correctly compared to Claude’s 21. In general, both models perform well on tasks with directly available information (92.6%/77.8%) and in reasoning on tasks for which the model contains insufficient information (100%/95.5%). As to expect, tasks with indirectly available information are most difficult as they require the model to retrieve and aggregate specific results.

We further compare the performance against the results reported by Hellin et al. (2025) on Sonnet 3.5. It is to note that the results are not directly comparable, because our system was only evaluated on a subset of IFC-Bench-v1, whereas Hellin et al. (2025) evaluate on the whole dataset. Although the functions of our MCP4IFC system are not adapted on the IFC-bench-v1 dataset tasks, the performance is similar to their system, which achieved an accuracy of 80%.

The tools mostly used by Sonnet 4.5 on correctly solved tasks can be seen in Table 3. Notably, the `execute_ifc_code_tool` function is called most often, which allows the model to write its own code. This is often necessary to aggregate information, e.g. when requiring spatial relationships, counting or selecting objects of a distinct type. Furthermore, this presents an important benefit over the tools defined by Hellin et al. (2025), as it allows the system to create its own task-specific tools to answer a query.

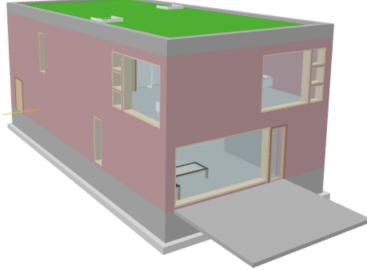


Figure 4: Duplex House

Q: Which rooms have direct access to outdoor spaces?
A: [...] 4 Rooms have direct access to outdoor spaces: A101, B101, A102, B102. [...]
Q: Are the walls of room A203 load bearing?
A: [...] The interior partition and plumbing walls in room A203 are NOT load-bearing [...]
Q: What is the depth of the foundation?
A: [...] The depth of the foundation is 0.3 meters (300 mm). [...]
Q: Are all internal doors compliant with accessibility requirements?
A: Based on my analysis of all 14 doors in your IFC project, not all internal doors are compliant with accessibility requirements. [...]

Figure 5: Questions and correctly given IFC-based answers by Sonnet 4.5 on the ARC Duplex using MCP4IFC.

Category	GPT-5 mini			Sonnet 4.5			Sonnet 3.5*
	Correct	Incorrect	Accuracy	Correct	Incorrect	Accuracy	Accuracy
Direct	25	2	92.6%	21	6	77.8%	95%
Indirect	7	9	43.8%	7	9	43.8%	62%
Insufficient	22	0	100.0%	21	1	95.5%	73%
Total	54	11	83.1%	49	16	75.4%	80%

Table 2: Comparison of scene querying results between **GPT-5 mini**, **Sonnet 4.5**, and **Sonnet 3.5**.

*Results for Sonnet 3.5 are reported by Hellin et al. (2025), evaluated on a larger dataset and therefore not directly comparable.

Our evaluation comes with important limitations. First, the binary correct/incorrect metric can be insufficient for certain ambiguous questions where an answer might be partially correct. In our evaluation, we optimistically marked such cases as correct, suggesting our reported accuracies represents an upper bound. A more nuanced evaluation scheme may be necessary for future work. Second, grouping tasks into the proposed categories by Hellin et al. (2025) led to difficulties, as, e.g., tasks with indirectly available information often contain a high proportion subtasks where direct retrieval is required, and vice versa, making a clean categorization difficult. However, we grouped tasks to the best of our knowledge and – as similar performance is achieved compared to

tool	count
<code>execute_ifc_code_tool</code>	91
<code>get_ifc_scene_overview</code>	45
<code>get_scene_info</code>	29
<code>get_object_info</code>	16
<code>execute_blender_code</code>	7
<code>search_ifc_knowledge</code>	2
<code>get_door_properties</code>	1
<code>capture_blender_3dviewport_screenshot</code>	1

Table 3: Distribution of tool usage among the correct solutions.

Table 4: The eight prompts used for the IFC editing experiments via dynamic code generation. Claude Sonnet 4.5 completed all tasks using the MCP server. The RAG column indicates whether the LLM invoked the retrieval tool.

Prompt	Input IFC	RAG	Outcome
1. Rename all selected walls to include their height in the name (e.g., “Wall-3.0m”).	small_house.ifc	No	Success
2. Change the description of the building to “High-rise residential tower”.	small_house.ifc	No	Success
3. Update the name of all <code>IfcDoor</code> objects by appending their floor level to the existing name.	small_house.ifc	No	Success
4. Add a custom property set “Thermal_Properties” to all walls with properties: U-value = 0.25, Insulation_Type = “Mineral Wool”.	small_house.ifc	Yes	Success
5. Find all slabs and add a “Fire_Rating” property with value “2HR”.	small_house.ifc	Yes	Success
6. Classify all walls and slabs in the IFC model using Uniclass 2015 (walls: Ss_25_10_20, slabs: Ss_25_30).	small_house.ifc	Yes	Success
7. Add cost information to all windows (unit cost \$500) and compute the total based on the window count.	small_house.ifc	No	Success
8. Update the owner history for selected elements with current timestamp and user “BIM Manager”.	small_house.ifc	Yes	Success

Hellin et al. (2025) – think that our assessment is close to the unreleased ground truth. Lastly, as the MCP4IFC system offers a variety of different tools to query and generate BIM structures, the system currently requires a large context size in the input prompt. As this directly represents a higher cost and less than 10 of the tools were actually required to successfully solve tasks in scene querying (see Table 3), it is advisable to generally disable generative tools for such a task and vice versa.

4.2 EDITING AND GENERATION EXPERIMENTS

In addition to the querying benchmark, we evaluated how MCP4IFC performs on editing and generation tasks. These experiments test whether the system can follow high-level instructions and perform modeling operations without manual GUI interaction. The experiments cover two main tasks: (1) semantic IFC editing via code generation, where the LLM writes and executes Python code to modify IFC data, and (2) IFC geometry generation, where natural language prompts are translated into full 3D building elements. All experiments were conducted using the Claude Sonnet 4.5 model in the Claude Desktop interface.

4.2.1 IFC EDITING VIA CODE GENERATION (SEMANTIC/NON-GEOMETRIC)

When predefined tools cannot complete a request, the LLM generates new Python code to perform the task (as described in Section 3.3). This experiment focuses on semantic and non-geometric IFC modifications, for which there are no specialized tools. We designed eight editing tasks (Table 4) that included batch renaming, creating property sets, classifying elements, and performing custom calculations. We prompted the LLM to perform these operations using our MCP framework and analyzed whether it could complete the task.

All experiments were performed on the `small_house.ifc` model from the BIM-Whale repository (Wisén, 2020). Claude Sonnet 4.5 completed all eight editing tasks via dynamic code generation. Each result was manually verified by comparing the IFC file before and after execution. Although the task set is limited, the results show that the LLM can reliably execute semantic edits and generate valid IFC-compatible code. More complex cases may fail when geometry, spatial context, or element dependencies are required. Overall, these experiments demonstrate that direct IFC editing via LLM-driven code generation is feasible for a range of semantic operations.

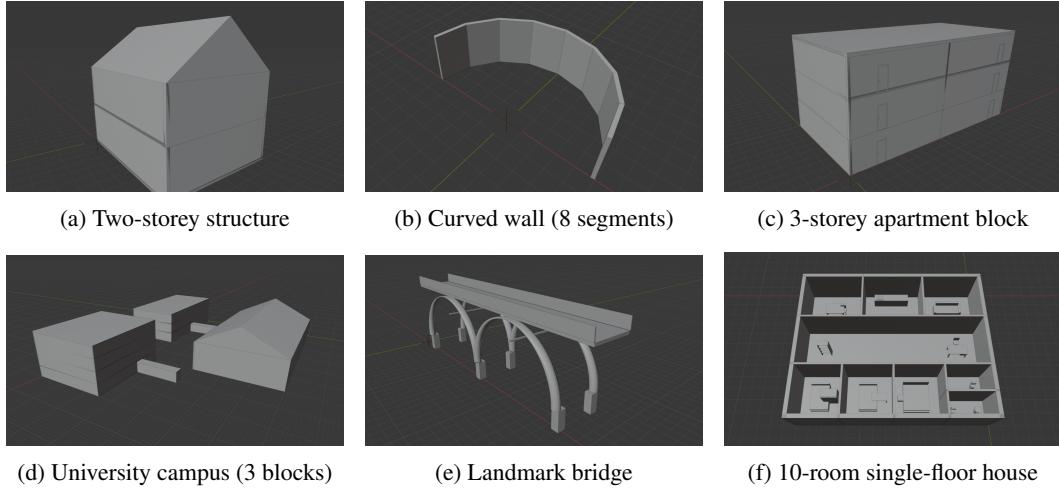


Figure 6: IFC object generation from textual prompts. Examples (a-f) are generated using the tools available in the MCP server. Here are the detailed prompts: (a) Build a two-story structure: ground floor slab 10 m × 8 m, four perimeter walls 3 m high, second floor slab at 3 m, another set of walls 3 m high, and a gable roof. (b) Create a curved wall approximation with 8 segments forming a semi-circle, radius 5 m, height 3 m. (c) Create a three-storey residential apartment block with two flats per floor, each flat having a living room, kitchen, bedroom, and bathroom. (d) Create a university campus with three buildings: a main lecture block, a library, and a sports hall connected by covered walkways. (e) Generate a landmark bridge structure with arch supports and pedestrian deck. (f) Generate a single-floor house with ten rooms, each furnished with appropriate interior objects (e.g., bed, table, or sofa).

4.2.2 IFC GEOMETRY GENERATION

We evaluated the model’s ability to generate IFC geometry both from single prompts and through multi-turn interactive workflows. Geometry creation in MCP4IFC is supported by predefined high-level tools 3.2.2 and by dynamic code generation 3.3. As shown in Figure 6, the system can create a variety of structures from textual descriptions, demonstrating that the model can interpret high-level design intent and translate it into valid IFC geometry. We further analyzed sequential design workflows in which the LLM iteratively refined a single IFC model across multiple user instructions. Figure 7 shows an example where the model preserved spatial context over six consecutive prompts to produce a coherent layout. This indicates strong potential for conversational BIM workflows where users can iteratively shape a design through natural language instead of manual GUI operations.

However, a closer review of the generated IFC files reveals some issues. In the example (f) shown in Figure 7, the geometry appears structurally valid, but the underlying IFC semantics were not fully specified. Walls did not connect properly at intersections, openings for windows and doors were not properly added, and rooms lacked `IfcSpace` definitions. Elements often lacked property sets and material assignments, and furniture lacked spatial containment relationships. These issues show that current models can generate geometric elements but struggle to maintain the relational and semantic structure required for reliable BIM workflows.

Overall, the results demonstrate that LLM-driven IFC generation is feasible but not yet semantically robust. Improving spatial reasoning, semantic consistency, and dependency handling remains essential. Future work should focus on fixing incomplete IFC structures and training models with better exposure to well-structured IFC data so they can learn to create better BIM models.

5 CONCLUSION & FUTURE WORK

In this work, we presented MCP4IFC, an MCP server for generating, editing, and querying IFC models built on top of the IfcOpenShell API. The framework connects large language models (LLMs)

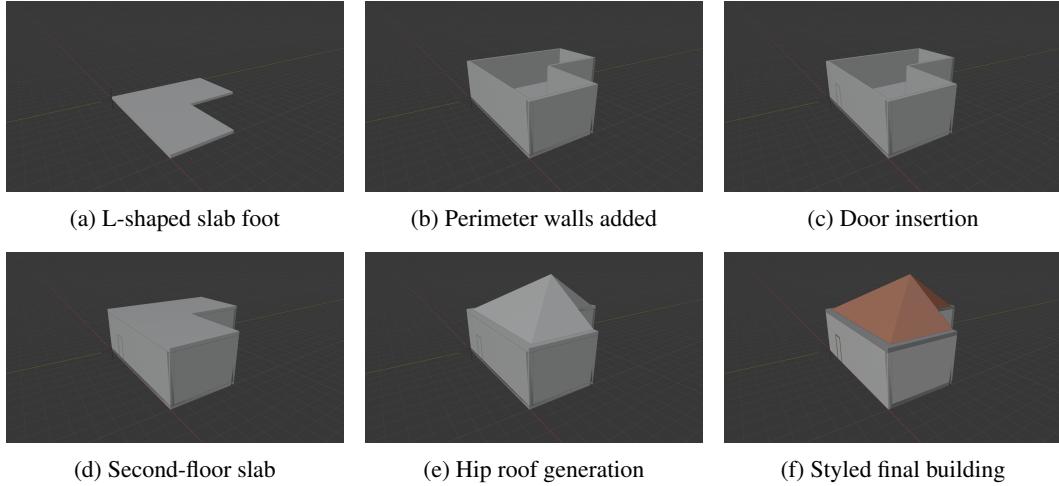


Figure 7: Step-by-step generation of an L-shaped building through sequential instructions. Here are the detailed prompts: (a) Create an L-shaped floor slab using polyline: vertices at (0,0), (10,0), (10,5), (5,5), (5,10), (0,10), thickness 0.25m at height 0. (b) Create walls along this L-shaped perimeter: height 3.5m, thickness 0.25m, following the slab outline. (c) Add a door at position (2, 0, 0) - entrance on the south side. (d) Create another L-shaped slab at height 3.5m matching the ground floor shape. (e) Add a hip roof covering the entire L-shaped building with 30-degree slope. Generate a landmark bridge structure with arch supports and pedestrian deck. (f) Can you style the house. Generate a single-floor house with ten rooms, each furnished with appropriate interior objects (e.g., bed, table, or sofa).

to BIM workflows through standardized MCP tool calls, enabling direct interaction with IFC data in an interoperable and transparent way. By combining predefined tool interfaces with dynamic code generation and retrieval-augmented learning, the system allows LLMs to perform a wide range of modeling and information retrieval tasks without depending on proprietary software APIs. Our experiments demonstrate that the proposed framework is capable of performing realistic BIM operations, from IFC querying to semantic editing and procedural geometry generation. While these experiments are not intended as comprehensive benchmarks, they demonstrate that the MCP4IFC framework effectively enables LLMs to perform end-to-end IFC manipulation through a standardized, extensible tool interface. MCP4IFC is released as open source to support further research in AI-assisted BIM design. We believe it provides a solid foundation for developing and evaluating future systems that integrate language models, spatial reasoning, and open standards to advance automation in the built environment. Concretely, various research directions can be derived from this:

Expanded Tool Library: The current toolset can be further extended to support higher-level modeling operations. New tools for common building components, such as staircases, elevators, and facades, would allow AI agents to complete complex design actions with a single tool call. In addition, community experts could contribute their own extensions, which would become immediately accessible to any connected AI client through the standardized MCP interface. Beyond predefined modeling tools, generative design capabilities can also be integrated. For instance, a floor-plan understanding model (Wei et al., 2025; Fan et al., 2022) could transform images or sketches into room layouts, while advanced mesh-generation algorithms (Wang et al., 2024a; Hao et al., 2024) could be combined with an LLM to convert complex geometries into structured IFC elements. Together, these additions would enable model-based design exploration within IFC-compliant workflows.

Agentic BIM Workflows and AI Model Improvement The current framework supports individual modeling actions, but full automation requires higher-level planning and decision-making. Future research could develop BIM-specific agents that can plan complete tasks, monitor progress, and react to feedback. Improving the underlying LLMs is also essential. A model fine-tuned on BIM data, tool-call traces, and code execution logs could reason more effectively about geometry and semantics. Such a domain-specialized model could run locally to reduce cost and latency. Together,

these advances would allow the agent to autonomously design, check, and refine IFC models through long-horizon reasoning. In addition, current LLMs and VLMs still face fundamental limitations in geometric and topological understanding (Yin et al., 2025; Mao et al., 2025). Future research could explore multi-modal spatial feedback mechanisms, allowing models to reason about geometry through structured representations such as graphs, point clouds, and visual embeddings (Bolya et al., 2025; Assran et al., 2025; Zhou et al., 2024). Integrating these world-model concepts from robotics could enhance spatial awareness and enable more reliable generative design, collision detection, and geometric reasoning in BIM workflows.

Extending to other BIM software: Most tools in the current framework are implemented through IfcOpenShell, with Bonsai and Blender serving as lightweight interfaces that provide a 3D modeling workspace and interactive editing capabilities. Because the core functionality is built on IfcOpenShell, these tools can be easily adapted to any BIM software environment that supports IFC import, modification, or visualization. This design makes the framework highly flexible and straightforward to integrate into diverse BIM platforms.

6 ACKNOWLEDGEMENTS

We would like to thank JotaDeRodriguez for valuable discussions and support regarding the MCP server integration, and Sylvain Hellin for providing the IFC-Bench-v1 dataset and for the insightful exchange. We also acknowledge the IfcOpenShell community for developing and maintaining the open platform on which this work builds. This research was supported by the Federal Ministry of Research, Technology and Space via the Show2Instruct project (Grant No. 01MK25008D).

REFERENCES

- Ahujasid. blender-mcp: Blender model context protocol integration. <https://github.com/ahujasid/blender-mcp>, 2025. Accessed: 2025-10-10.
- Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Mojtaba, Komeili, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, Sergio Arnaud, Abha Gejji, Ada Martin, Francois Robert Hogan, Daniel Dugas, Piotr Bojanowski, Vasil Khalidov, Patrick Labatut, Francisco Massa, Marc Szafraniec, Kapil Krishnakumar, Yong Li, Xiaodong Ma, Sarath Chandar, Franziska Meier, Yann LeCun, Michael Rabbat, and Nicolas Ballas. V-jepa 2: Self-supervised video models enable understanding, prediction and planning, 2025. URL <https://arxiv.org/abs/2506.09985>.
- Autodesk Revit. Bim software overview. <https://www.autodesk.com/de/products/revit/overview>, 2025. Accessed: 2025-10-21.
- Daniel Bolya, Po-Yao Huang, Peize Sun, Jang Hyun Cho, Andrea Madotto, Chen Wei, Tengyu Ma, Jiale Zhi, Jathushan Rajasegaran, Hanoona Rasheed, Junke Wang, Marco Monteiro, Hu Xu, Shiyu Dong, Nikhila Ravi, Daniel Li, Piotr Dollár, and Christoph Feichtenhofer. Perception encoder: The best visual embeddings are not at the output of the network, 2025. URL <https://arxiv.org/abs/2504.13181>.
- Bonsai. Ifcopenshell / bonsai - beautiful, detailed, and data-rich openbim. <https://bonsaibim.org/>, 2025. Accessed: 2025-10-10.
- Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS ’20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.
- William Chen, Siyi Hu, Rajat Talak, and Luca Carlone. Extracting zero-shot common sense from large language models for robot 3d scene understanding, 2022. URL <https://arxiv.org/abs/2206.04585>.

- Dawson-Haggerty et al. trimesh, 2019. URL <https://trimesh.org/>.
- Zihan Deng, Changyu Du, Stavros Nousias, and André Borrmann. Bimgent: Towards autonomous building modeling via computer-use agents. *arXiv preprint arXiv:2506.07217*, 2025.
- Yaxian Dong, Zijun Zhan, Yuqing Hu, Daniel Mawunyo Doe, and Zhu Han. Ai bim coordinator for non-expert interaction in building design using llm-driven multi-agent systems. *Automation in Construction*, 180:106563, 2025.
- Changyu Du, Sebastian Esser, Stavros Nousias, and André Borrmann. Text2bim: Generating building models using a large language model-based multi-agent framework. *arXiv preprint arXiv:2408.08054*, 2024a.
- Changyu Du, Stavros Nousias, and André Borrmann. Towards a copilot in bim authoring tool using a large language model-based agent for intelligent human-machine interaction. *arXiv preprint arXiv:2406.16903*, 2024b.
- Jayakrishna Duggempudi, Lu Gao, Ahmed Senouci, Zhe Han, and Yunpeng Zhang. Text-to-layout: A generative workflow for drafting architectural floor plans using llms. *arXiv preprint arXiv:2509.00543*, 2025.
- Tim Engelbracht, René Zurbrügg, Marc Pollefey, Hermann Blum, and Zuria Bauer. Spotlight: Robotic scene understanding through interaction and affordance detection, 2024. URL <https://arxiv.org/abs/2409.11870>.
- Zhiwen Fan, Tianlong Chen, Peihao Wang, and Zhangyang Wang. Cadtransformer: Panoptic symbol spotting transformer for cad drawings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10986–10996, June 2022.
- David Fernandes, Sahej Garg, Matthew Nikkel, and Gursans Guven. A gpt-powered assistant for real-time interaction with building information models. *Buildings*, 14(8):2499, 2024.
- Graphisoft. Archicad — plans & products. <https://www.graphisoft.com/plans-and-products/archicad/>, 2025. Accessed: 2025-10-21.
- Peizhuo Guo, Huiyuan Xue, Jun Ma, and Jack CP Cheng. Arcbim: Low-prerequisite, high-flexible, and cost-effective bim information retrieval with condensed multi-llm agent-based dsl code generation. *High-Flexible, and Cost-Effective Bim Information Retrieval with Condensed Multi-Llm Agent-Based Dsl Code Generation*, 2025.
- Zekun Hao, David W. Romero, Tsung-Yi Lin, and Ming-Yu Liu. Meshtron: High-fidelity, artist-like 3d mesh generation at scale. *ArXiv*, abs/2412.09548, 2024. URL <https://api.semanticscholar.org/CorpusID:274656066>.
- Sylvain Hellin, Stavros Nousias, and André Borrmann. Natural language information retrieval from bim models: An llm-based agentic workflow approach. In *2025 European Conference on Computing in Construction (CIB W78 Conference on IT in Construction)*, Porto, Portugal, 2025.
- Yining Hong, Haoyu Zhen, Peihao Chen, Shuhong Zheng, Yilun Du, Zhenfang Chen, and Chuang Gan. 3d-llm: Injecting the 3d world into large language models, 2023. URL <https://arxiv.org/abs/2307.12981>.
- Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. Model context protocol (mcp): Landscape, security threats, and future research directions. *arXiv preprint arXiv:2503.23278*, 2025.
- IfcOpenShell. Open-source toolkit for working with ifc data. <https://ifcopenshell.org/>, 2025. Accessed: 2025-10-10.
- ISO 16739-1:2024. Industry foundation classes (ifc) for data sharing in the construction and facility management industries — part 1: Data schema, March 2024. URL <https://www.iso.org/standard/84123.html>. Accessed: 2025-10-10.
- Suhhyung Jang, Ghang Lee, Jiseok Oh, Junghun Lee, and Bonsang Koo. Automated detailing of exterior walls using nadia: Natural-language-based architectural detailing through interaction with ai. *Advanced Engineering Informatics*, 61:102532, 2024.

- Baoxiong Jia, Yixin Chen, Huangyue Yu, Yan Wang, Xuesong Niu, Tengyu Liu, Qing Li, and Siyuan Huang. Sceneverse: Scaling 3d vision-language learning for grounded scene understanding, 2024. URL <https://arxiv.org/abs/2401.09340>.
- JotaDeRodriguez. Bonsai_mcp: Model context protocol integration for ifc via bonsai and blender. https://github.com/JotaDeRodriguez/Bonsai_mcp, 2025. Accessed: 2025-10-10.
- Mohammad Sadil Khan, Sankalp Sinha, Talha Uddin, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2cad: Generating sequential cad designs from beginner-to-expert level text prompts. *Advances in Neural Information Processing Systems*, 37:7552–7579, 2024.
- Sicong Leng, Yangqiaoyu Zhou, Mohammed Haroon Dupty, Wee Sun Lee, Sam Joyce, and Wei Lu. Tell2design: A dataset for language-guided floor plan generation. *ArXiv*, abs/2311.15941, 2023. URL <https://api.semanticscholar.org/CorpusID:259370678>.
- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS ’20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.
- Ang Li, Peter Kok-Yiu Wong, Xingyu Tao, Jun Ma, and Jack CP Cheng. An interactive system for 3d spatial relationship query by integrating tree-based element indexing and llm-based agent. *Advanced Engineering Informatics*, 66:103375, 2025a.
- Rong Li, Shijie Li, Lingdong Kong, Xulei Yang, and Junwei Liang. Zero-shot 3d visual grounding from vision-language models, 2025b. URL <https://arxiv.org/abs/2505.22429>.
- Bingru Liu and Hainan Chen. Bimcoder: A comprehensive large language model fusion framework for natural language-based bim information retrieval. *Applied Sciences*, 15(14):7647, 2025.
- Antonello Longo, Chanyoung Chung, Matteo Palieri, Sung-Kyun Kim, Ali Agha, Cataldo Guaragnella, and Shehryar Khattak. Pixels-to-graph: Real-time integration of building information models and scene graphs for semantic-geometric human-robot understanding, 2025. URL <https://arxiv.org/abs/2506.22593>.
- Wufei Ma, Luoxin Ye, Celso M de Melo, Jieneng Chen, and Alan Yuille. Spatialllm: A compound 3d-informed design towards spatially-intelligent large multimodal models, 2025. URL <https://arxiv.org/abs/2505.00788>.
- Yongsen Mao, Junhao Zhong, Chuan Fang, Jia Zheng, Rui Tang, Hao Zhu, Ping Tan, and Zihan Zhou. Spatialllm: Training large language models for structured indoor modeling, 2025. URL <https://arxiv.org/abs/2506.07491>.
- Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020. URL <https://arxiv.org/abs/2003.08934>.
- Model Context Protocol. Introduction. <https://modelcontextprotocol.io/docs/getting-started/intro>, 2025. Accessed: 2025-10-10.
- Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. *ArXiv*, abs/2305.15334, 2023. URL <https://api.semanticscholar.org/CorpusID:258865184>.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessí, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: language models can teach themselves to use tools. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS ’23, Red Hook, NY, USA, 2023. Curran Associates Inc.

- Oliver Schulz, Jeroen Werbrouck, and Jakob Beetz. Towards scene graph descriptions for spatial representations in the built environment. In *30th International Workshop on Intelligent Computing in Engineering, EG-ICE*, volume 2023, 2023.
- Bilal Succar. Building information modelling framework: A research and delivery foundation for industry stakeholders. *Automation in Construction*, 18(3):357–375, 2009. ISSN 0926-5805. doi: <https://doi.org/10.1016/j.autcon.2008.10.003>. URL <https://www.sciencedirect.com/science/article/pii/S0926580508001568>.
- Vectorworks. 3d design software for bim, cad & modelling. <https://www.vectorworks.net/>, 2025. Accessed: 2025-10-21.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.
- Zhengyi Wang, Jonathan Lorraine, Yikai Wang, Hang Su, Jun Zhu, Sanja Fidler, and Xiaohui Zeng. Llama-mesh: Unifying 3d mesh generation with language models. *ArXiv*, abs/2411.09595, 2024a. URL <https://api.semanticscholar.org/CorpusID:274023341>.
- Zora Zhiruo Wang, Akari Asai, Xinyan Velocity Yu, Frank F. Xu, Yiqing Xie, Graham Neubig, and Daniel Fried. Coderag-bench: Can retrieval augment code generation? In *North American Chapter of the Association for Computational Linguistics*, 2024b. URL <https://api.semanticscholar.org/CorpusID:270620678>.
- Xinguang Wei, Haomin Wang, Shenglong Ye, Ruifeng Luo, Yanting Zhang, Lixin Gu, Jifeng Dai, Yu Qiao, Wenhui Wang, and Hongjie Zhang. Point or line? using line-based representation for panoptic symbol spotting in cad drawings. *ArXiv*, abs/2505.23395, 2025. URL <https://api.semanticscholar.org/CorpusID:278996383>.
- André Wisén. bim-whale-ifc-samples: Ifc sample files for the bim whale project. GitHub repository, 2020. URL <https://github.com/andrewisen/bim-whale-ifc-samples/tree/main>. Accessed: 2025-10-23.
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024.
- Jianing Yang, Xuweiyi Chen, Nikhil Madaan, Madhavan Iyengar, Shengyi Qian, David F. Fouhey, and Joyce Chai. 3d-grand: A million-scale dataset for 3d-llms with better grounding and less hallucination. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 29501–29512, June 2025a.
- John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: agent-computer interfaces enable automated software engineering. In *Proceedings of the 38th International Conference on Neural Information Processing Systems, NIPS '24*, Red Hook, NY, USA, 2025b. Curran Associates Inc. ISBN 9798331314385.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.
- Baiqiao Yin, Qineng Wang, Pingyue Zhang, Jianshu Zhang, Kangrui Wang, Zihan Wang, Jieyu Zhang, Keshigeyan Chandrasegaran, Han Liu, Ranjay Krishna, Saining Xie, Manling Li, Jiajun Wu, and Li Fei-Fei. Spatial mental modeling from limited views, 2025. URL <https://arxiv.org/abs/2506.21458>.
- Zeqing Yuan, Haoxuan Lan, Qiang Zou, and Junbo Zhao. 3d-premise: Can large language models generate 3d shapes with sharp features and parametric control? *arXiv preprint arXiv:2401.06437*, 2024.
- Chenyangguang Zhang, Alexandros Delitzas, Fangjinhua Wang, Ruida Zhang, Xiangyang Ji, Marc Pollefeys, and Francis Engelmann. Open-vocabulary functional 3d scene graphs for real-world indoor spaces, 2025a. URL <https://arxiv.org/abs/2503.19199>.

Jiahui Zhang, Yurui Chen, Yanpeng Zhou, Yueming Xu, Ze Huang, Jilin Mei, Junhui Chen, Yu-Jie Yuan, Xinyue Cai, Guowei Huang, Xingyue Quan, Hang Xu, and Li Zhang. From flatland to space: Teaching vision-language models to perceive and reason in 3d. *ArXiv*, abs/2503.22976, 2025b. URL <https://api.semanticscholar.org/CorpusID:277510935>.

Junwen Zheng and Martin Fischer. Dynamic prompt-based virtual assistant framework for bim information search. *Automation in Construction*, 155:105067, 2023.

Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. Dino-wm: World models on pre-trained visual features enable zero-shot planning. *arXiv preprint arXiv:2411.04983*, 2024.

Shuyan Zhou, Uri Alon, Frank F. Xu, Zhiruo Wang, Zhengbao Jiang, and Graham Neubig. Docprompting: Generating code by retrieving the docs. In *International Conference on Learning Representations*, 2022. URL <https://api.semanticscholar.org/CorpusID:252734952>.

A APPENDIX

A.1 DYNAMIC CODE GENERATION EXAMPLE

This section provides an example of a script generated by the LLM during the semantic (non-geometric) IFC editing experiments described in Section 4.2.1. When a required operation is not covered by predefined tools, the LLM writes Python code that is executed inside the secure Blender sandbox. Listing 1 shows a script generated by Claude Sonnet 4.5 to update the description of the building entity. Before writing the code, the model may call scene-query tools. These tools provide additional context, such as the ID of selected objects and the scene overview. After it has enough information, the model generates the code and uses the `execute_ifc_code_tool()` to execute the generated script. The script first loads the current IFC file that is already open in the Bonsai Blender environment using the `get_ifc_file()` function. Then it performs the operation and uses the `save_and_load_ifc()` function to write the changes back to the IFC file and refreshes the model. This shows how the system can edit IFC data through code without any GUI actions.

```

import ifcopenshell
import ifcopenshell.api
from blender_addon.api.ifc_utils import get_ifc_file,
    save_and_load_ifc

# Get the IFC file
ifc_file = get_ifc_file()
# Find the building by GUID
building_guid = "3UdjywU2L4v9tTcFvuqwGm"
building = ifc_file.by_guid(building_guid)
# Check current description
print(f"Building name: {building.Name}")
print(f"Current description: {building.Description}")
# Edit the building's description attribute
ifcopenshell.api.run(
    "attribute.edit_attributes",
    ifc_file,
    product=building,
    attributes={"Description": "High-rise residential tower"})
# Verify the change
print(f"New description: {building.Description}")
# Save the changes
save_and_load_ifc()
print("Building description successfully updated!")

```

Listing 1: LLM-generated script to change the description of the building to 'High-rise residential tower' (executed in Blender sandbox).

```
I will get the current scene information for you.
<<tool_call: get_scene_info>>

Scene Summary
The current Blender scene contains a basic IFC project structure
with 9 objects total:
IFC Spatial Structure
- IfcProject - My Project (root container)
- IfcSite - My Site
- IfcBuilding - My Building
- IfcBuildingStorey - My Storey (active floor level)
Building Elements
- 4 Walls forming a rectangular enclosure:
  - Wall_001 at origin (0, 0, 0)
  - Wall_002 at (10, 0, 0)
  - Wall_003 at (10, 10, 0)
  - Wall_004 at (0, 10, 0)
  These appear to create a 10m x 10m room layout.
Type Objects
- IfcWallType - wall (template/type definition, currently hidden)

All objects are currently visible except for the wall type
definition. No objects are currently selected. The scene
represents a simple rectangular room structure within a
standard IFC hierarchical organization.
```

Listing 2: The LLM generated output for the prompt - can you give a summary of the existing scene, use the `get_scene_info` function. The model calls the tool (« `tool_call`») and then summarizes the tool's raw JSON output.

A.2 SCENE QUERY OUTPUT EXAMPLE

To demonstrate the LLM’s scene-understanding capability, we created a simple test scene in an IFC file containing four walls in a 10x10m grid. The LLM was then prompted to summarize this scene. Listing 2 shows the LLM output and Listing 3 shows the raw JSON data retrieved by the LLM’s tool call.

A.3 FLOOR PLAN GENERATION FROM IMAGES

This experiment explores whether the multimodal abilities of LLMs could be used for floor plan understanding. Three floor-plan images from the Tell2Design dataset (Leng et al., 2023) were provided separately, and the LLM (Claude Sonnet 4.5) was prompted to generate the corresponding wall layouts. The input and output are shown in Figure 8. The LLM captured the overall structure but struggled with the spatial organization. Fine-grained geometry and more intricate layout features were often missing or incorrectly interpreted. These suggest that while multimodal LLMs can extract high-level spatial features from architectural drawings, they currently lack the precision needed for accurate reconstruction. Integrating a specialized floor-plan recognition model (Fan et al., 2022; Wei et al., 2025) could improve performance by providing a more reliable extraction of boundaries and room structures.

A.4 VISUAL FEEDBACK CHECK USING SCREENSHOT TOOLS

To explore whether a multimodal LLM can interpret BIM scenes directly from images, we ran a small experiment. The goal of this experiment was to assess whether the model could understand the 3D scene shown in each image and support basic visual reasoning tasks. We captured screenshots from different IFC models and asked the model a set of questions about visible elements. These questions can be categorized into three common types:

```
{
  "count": 9,
  "total": 9,
  "offset": 0,
  "limit": 9,
  "objects": [
    {
      "name": "IfcProject/My Project",
      "type": "EMPTY",
      "location": [0.0, 0.0, 0.0],
      "visible": true,
      "selected": false,
      "guid": "0CsEa6q1jESwVHd0H5grP1",
      "ifc_class": "IfcProject"
    },
    {
      "name": "IfcBuildingStorey/My Storey",
      "type": "EMPTY",
      "location": [0.0, 0.0, 0.0],
      "visible": true,
      "selected": false,
      "guid": "3POxSr5YnFlhRhdFQxAIXX",
      "ifc_class": "IfcBuildingStorey"
    },
    {
      "name": "IfcWall/Wall_001",
      "type": "MESH",
      "location": [0.0, 0.0, 0.0],
      "visible": true,
      "selected": false,
      "guid": "3Cqrj0WO5Erf3j77QECM8D",
      "ifc_class": "IfcWall"
    },
    {
      "name": "IfcWall/Wall_002",
      "type": "MESH",
      "location": [10.0, 0.0, 0.0],
      "visible": true,
      "selected": false,
      "guid": "23EPa7DQvF2foQfEeTNPD_",
      "ifc_class": "IfcWall"
    },
    ....
  ]
}
```

Listing 3: Raw JSON output from the `get_scene_info` tool call. This data structure lists all scene objects, including their properties and unique GUIDs. Note that even in a simple 4-wall scene, the output includes hierarchical elements (e.g., IfcProject, IfcBuildingStorey) as separate objects, since they have their own GUIDs. The output is truncated here.

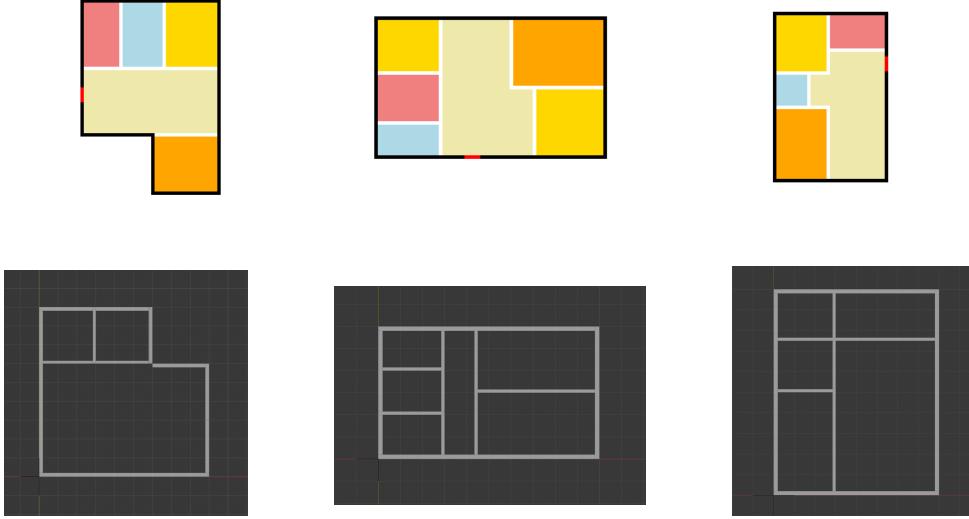


Figure 8: Floor plan generation using a multimodal LLM. The top row shows input floor plans from the Tell2Design dataset , and the bottom row shows the corresponding wall layouts generated by the MCP server.

- **Counting (25 questions):** Questions regarding the number of elements, such as walls, chairs, windows, or other elements, found in BIM.
- **Spatial Relationships (9 questions):** Questions about the relative positioning of objects, e.g., "Where is the TV located relative to the sofa?"
- **Attribute Queries (5 questions):** Questions regarding object attributes like color, material, or location.

In total, 40 questions were evaluated. Table 5 shows a few examples of the image-based questions, and the corresponding screenshots are presented in Figure 9. For evaluation, we used a three-level scoring system: 1.0 for correct answers, 0.5 for partially correct answers, and 0 for incorrect answers. Using this metric, Claude Sonnet 4.5 achieved an overall score of **73%**. The model was often able to recognize major objects and answer attribute questions correctly. However, its performance on counting was less consistent, and some spatial relationships were incorrect. These results suggest that while the model can extract high-level visual information from BIM images, its understanding of scene geometry and the quantity of objects remains limited. Nonetheless, this simple experiment shows that screenshot-based visual feedback can already support certain interactions, and motivates stronger vision-language integration for future BIM applications.



Figure 9: Screenshots used for the visual question answering experiment. The first four images are shown in a 2×2 layout (top-left: P1, top-right: P2, bottom-left: P3, bottom-right: P4), followed by P5 and P6).

Table 5: A few example questions for different building scenes.

Picture No.	Spatial Question
P1	How many chairs are there in the image?
P1	Where can you find a pair of chairs in the image?
P1	What color are the chairs?
P2	Where is the TV located relative to the sofa?
P2	Is there a table in the room?
P2	What element in the scene is blue?
P3	How many pillows are there on the bed?
P3	What is in front of the chair?
P3	How many drawers does the table have?
P4	Where is the sink in relation to the stove?
P4	How many stoves are on the kitchen counter?
P4	How many doors do you count in the kitchen? Give an exact number.
P5/P6	How many floors does this building have?
P5/P6	How many windows are on this building?
P5/P6	How many doors are on this building?