

IFC-Copilot: A Tool-Based Framework for LLM-Driven IFC Building Design

Bharathi Kannan Nithyanantham¹, Tobias Sesterhenn², Ashwin Nedungadi¹,
Sergio Peral Garijo², Janis Zenkner², Christian Bartelt², Stefan Lüdtkke¹

¹ University of Rostock, ² Clausthal University of Technology

Correspondence to: bharathikannan.nithyanantham@uni-rostock.de, tobias.sesterhenn@tu-clausthal.de

Abstract. Integrating generative AI into Architecture, Engineering, and Construction (AEC) workflows requires systems that can translate natural language instructions into reliable actions on standardized building data models. We present an open-source, tool-based framework that enables Large Language Models (LLMs) to directly query, create, and edit Industry Foundation Classes (IFC) building models. The framework includes predefined parametric tools for creating and modifying common building elements, a dynamic code generation system for handling tasks beyond the fixed toolset, and retrieval-augmented generation (RAG) to support code generation using relevant IFC documentation. All tools are exposed through JSON schemas and can be invoked by any MCP-compatible LLM client, providing an extensible interface for IFC manipulation without relying on proprietary BIM APIs. Evaluated with Claude Sonnet 4.5 and GPT 5.4, the framework achieves up to 86% accuracy on a custom benchmark and 73.3% on the IFC-bench-v1 querying benchmark, showing significant improvements over baseline approaches. We also demonstrate multi-turn design workflows, analyze common failure modes, and release the framework as open source for research and development.

1. Introduction

Building Information Modeling (BIM) has transformed AEC workflows by enabling architects and project stakeholders to collaborate on a shared, information-rich digital representation instead of relying on disconnected 2D drawings (Succar, 2009). The Industry Foundation Classes (IFC) standard provides an open, vendor-neutral schema for BIM data exchange across different tools and organizations (ISO 16739-1, 2024). However, creating, editing, and maintaining BIM models remains complex and is typically handled by trained specialists. Recent progress in Large Language Models (LLMs) suggests a new way of interacting with software. LLMs can interpret natural language, plan sequences of actions, and invoke external tools to accomplish goals (Wang et al., 2023; Yao et al., 2023). In BIM, this opens up the idea of an AI assistant that accepts high-level instructions (e.g., add an entrance door to the south wall) and execute the corresponding IFC operations automatically.

Existing approaches have explored natural-language BIM creation and editing (Deng et al., 2025; Du et al., 2024a) and BIM querying (Hellin et al., 2025; Li et al., 2025), but most systems depend on proprietary BIM APIs (e.g., Revit or Vectorworks). These APIs do not operate directly on the IFC data structure and limit portability across BIM platforms. To address these limitations, we present a tool-based framework that lets LLMs work directly at the IFC layer to query, create, and edit models through structured tool calls. The framework builds on the open-source IfcOpenShell library¹ and the Bonsai add-on² in Blender. Because the interface operates directly on IFC, the same tools can be reused across different IFC-compatible platforms. We use the Model Context Protocol (MCP)³ as an interface layer

¹ <https://docs.ifcopenshell.org/>

² <https://bonsaibim.org/>

³ <https://modelcontextprotocol.io/docs/getting-started/intro>

to connect the tools implemented to external LLM agents. A key feature of the framework is its dynamic tool expansion. If a user request is not covered by the predefined tools, the LLM can generate and execute IFC-specific Python code at runtime. For better reliability, the system retrieves relevant IfcOpenShell documentation and schema snippets from a local vector store, which helps the model generate more accurate code through retrieval-augmented generation (Lewis et al., 2020; Brown et al., 2020).

The framework is designed for three types of users: BIM practitioners who want to interact with IFC models using natural language, researchers studying LLM-BIM integration, and developers building AI-assisted construction workflows who can extend the toolset with domain-specific operations. To our knowledge, this is the first tool-based framework of this scope that enables LLMs to query, create, and edit IFC models while also supporting dynamic code generation. Our main contributions are:

- An open-source framework with 52 BIM tools for querying, creating, editing, and generating IFC models through structured tool interfaces.
- A three-tier tool architecture combining predefined parametric tools, dynamic IFC code generation, and retrieval-augmented generation for extensibility.
- Quantitative evaluation on a custom IFC benchmark and IFC-bench-v1 with multiple LLMs, comparing our framework against baseline configurations with analysis of common failure modes.

2. Related Works

2.1 LLMs as Tool-Using Agents

Tool-augmented LLMs allow language models to use external tools through structured function calls. In this setup, the LLM first receives descriptions of the available tools. Each tool has a name, a short explanation, and a defined input and output format, usually written as a JSON schema. During inference, the model selects a suitable tool, generates the required inputs, and uses the returned results to decide the next step. ReAct (Yao et al., 2023) introduced a framework where an LLM repeatedly plans actions, calls tools, observes results, and continues reasoning based on the feedback. Toolformer (Schick et al., 2023) showed that LLMs can learn how and when to use APIs without task-specific training labels. Datasets like ToolBench (Qin et al., 2023) provide large collections of tool-use examples for training models. Major LLM providers support function calling through their own APIs, but these interfaces are often not compatible across different LLM systems. The Model Context Protocol (MCP) addresses this by defining a standardized interface for tool discovery and invocation. In MCP, a server exposes tools that compatible LLM systems can discover and use through a uniform JSON-RPC protocol. Our framework uses MCP as its transport layer to ensure compatibility across different LLM systems.

2.2 LLM-Based BIM Systems

LLMs have been applied to both BIM creation and editing and natural-language querying. For generation, most systems translate natural-language instructions into executable actions on proprietary BIM APIs. Text2BIM (Du et al., 2024a) and earlier work (Du et al., 2024b) use multiple LLM agents with custom tools built on the Vectorworks API. NADIA (Jang et al., 2024), Text-to-Layout (Duggempudi et al., 2025), Dong et al. (2025), and Fernandes et al. (2024) target the Revit API for tasks such as layout generation, wall creation, clash detection,

and information retrieval. BIMgent (Deng et al., 2025) takes a different approach, controlling BIM software through mouse and keyboard actions using screenshots as visual feedback. For querying, Zheng and Fischer (2023) convert BIM data into MongoDB and use an LLM to generate database operations, Hellin et al. (2025) rely on a predefined function library for IFC retrieval, Li et al. (2025) combine spatial indexing with multi-agent LLMs for complex spatial queries, ARCBIM (Guo et al., 2025) translates queries into executable code via specialized LLMs, and BIMCoder (Liu and Chen, 2025) maps queries to SQL-like or IFCQL statements. Most of these systems are tightly coupled to proprietary platforms such as Revit and Vectorworks, limiting portability across BIM environments. In contrast, our framework operates directly on the open IFC standard (ISO 16739-1, 2024), providing a reusable tool interface applicable across any BIM environment that supports IFC.

3. System Architecture

The framework follows the standard MCP architecture and consists of three components: an AI client that provides the LLM interface, an MCP server that exposes the implemented tools and routes requests, and a Blender add-on that executes BIM operations using IfcOpenShell. A detailed workflow is shown in Figure 1. First, the AI client connects to the MCP server and retrieves all the available tool definitions in JSON format. These tool definitions are inserted into the system prompt so the LLM can interpret their functionality through in-context learning. When a user submits a request, the LLM selects an appropriate tool, generates structured parameters, and sends the request to the MCP server. The server validates the request and forwards it through a socket connection to the Blender add-on, which executes the operation and returns a structured result. The LLM then decides the next step and continues until the task is completed or a client-side iteration limit is reached.

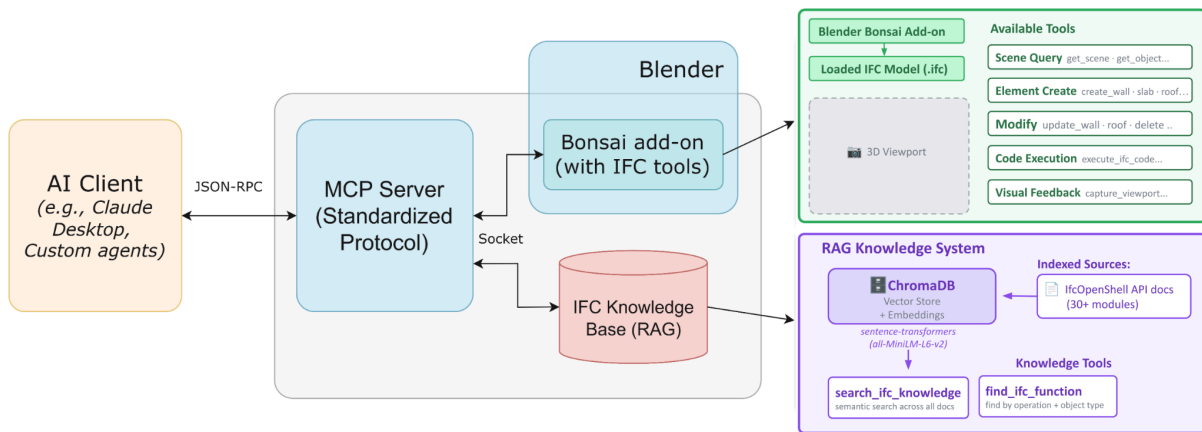


Figure 1: System architecture

3.1 Tool Interface Design

Each tool is defined by four components: (i) a unique name (e.g., create_wall), (ii) a natural-language description of its purpose and usage, (iii) an input schema in JSON Schema format with required parameters, types, and default values, and (iv) a structured JSON output containing a success status, created element identifiers such as IFC GlobalIds, and diagnostic information. These descriptions serve as in-context documentation, making tools self-describing and usable without additional model training. Adding new tools only requires providing their schema and descriptions in the system prompt. Table 1 summarizes the 52

tools in our framework organized by category. Figure 2 shows a multi-turn interaction example in which the LLM creates a rectangular room with a floor slab and also illustrates simplified schema representations as they appear in the system prompt during inference.

Table 1: Tool taxonomy with categories, counts, and examples.

Category	Count	Examples
Scene Query	11	get_scene_info, get_object_info, get_wall_properties, ..
Creation	18	create_wall, create_slab, create_roof, create_surface_style, ..
Modification	8	update_wall, update_roof, update_slab, ..
Geometry/Mesh	4	create_trimesh_ifc, create_mesh_ifc, ..
Code Execution + RAG	7	execute_ifc_code_tool, execute_blender_code, search_ifc_knowledge, find_ifc_function, ..
Visual Feedback	2	capture_blender_3dviewport, capture_blender_window
System Utilities	2	list_blender_commands, ensure_ifc_knowledge_ready

User

Create a rectangular room with 4 walls: 6m × 4m, walls 3m high, 0.2m thick

Assistant

I'll create 4 walls using `create_two_point_wall`, placing them at corners: (0,0) → (6,0) → (6,4) → (0,4) → back to (0,0).

▶ `create_two_point_wall` start=[0,0,0], end=[6,0,0], name="South", h=3.0, t=0.2

```
{"success": true, "wall_guid": "3Fk8x..."}
```

▶ Similar wall calls truncated here..

```
create_two_point_wall(...)
```

```
{"success": true, "wall_guid": ".."} (returned for each wall)
```

▶ `create_two_point_wall` start=[0,4,0], end=[0,0,0], name="West", h=3.0, t=0.2

```
{"success": true, "wall_guid": "1Ef6p..."}
```

I've created a 6m × 4m rectangular room with 4 walls (3m high, 0.2m thick). All walls were placed successfully.

User

Could you add a floor slab now?

Assistant

I'll create the floor slab now.

▶ `create_slab` polyline=[[0,0],[6,0],[6,4],[0,4]], depth=0.2

```
{"success": true, "slab_guid": "4Gh7q..."}
```

```
create_two_point_wall
{
  "name": "create_two_point_wall",
  "description": "Create a wall between two 3D points. The input arguments are (...), It returns (...) and Examples of how to use this function (...)",
  "input_schema": {
    "start_point": "[x, y, z]",
    "end_point": "[x, y, z]",
    "name": "string = Two Point Wall",
    "t": "float = 0.2",
    "h": "float = 3.0"
  },
  "output": {
    "success": "bool",
    "wall_guid": "string",
    "name": "string"
  }
}
```

```
create_slab
{
  "name": "create_slab",
  "description": "Create a parametric IFC slab with comprehensive geometric and material properties. The input arguments are (...), It returns (...) and Examples of how to use this function (...)",
  "input_schema": {
    "name": "string = New Slab",
    "polyline": "[[x1, y1], [x2, y2], ...]",
    "depth": "float = 0.2",
    "location": "[x, y, z]",
    "rotation": "[rx, ry, rz]",
    "geometry_properties": {
      "direction_sense": "str",
      "offset": "float",
      "x_angle": "float",
      "clippings": "list"
    }
  },
  "transformation_matrix": "4x4 matrix",
  "material": "string",
  "slab_type_guid": "string",
  "verbose": "bool = false"
},
"output": {
  "success": "bool",
  "slab_guid": "string",
  "message": "string",
  "properties": "object"
}
```

Figure 2: Example multi-turn interaction with claude sonnet 4.5 creating a rectangular room and a simplified representation of example tool definitions that are injected in the system prompt.

The framework provides a set of predefined tools that cover common IFC tasks across scene query, element creation and modification, property retrieval, style and material control, and visual feedback (Table 1). Scene query tools help the model inspect the current state of the

project before taking any action. Creation tools add common building elements such as walls, slabs, roofs, doors, windows, and stairs and returns a GlobalId that can be reused in subsequent tool calls. Modification tools allow the LLM to update existing elements by changing their parameters or geometry. Together, these tools cover a wide range of common BIM operations and encode correct IFC patterns such as spatial containment and geometric representations, so the LLM only needs to specify parameters.

For tasks beyond the predefined set, the framework adds a second layer combining mesh tools, code execution, and retrieval support. Mesh tools use in-context Trimesh examples (Dawson-Haggerty et al., 2019) to guide the LLM in generating freeform geometry (e.g., curved or irregular shapes), which is then converted into IFC elements. Dynamic code generation tools allow the LLM to write and execute Python code using IfcOpenShell during inference for custom IFC edits. The framework also provides in-context guidance and helper functions for interacting with the loaded model. Code generation is further supported by a retrieval system built on indexed IfcOpenShell documentation in a ChromaDB vector database, which helps ground the generated code in the correct API usage. This is especially useful when the API changes often. All generated code runs in a restricted sandbox with limited libraries, blocked file system and network access, and execution timeouts. Each run returns structured feedback, allowing the LLM to iteratively diagnose and correct failures.

3.2 Tool Interface Design

A key design consideration when developing these tools is how to convert existing software operations into interfaces that an LLM can use reliably. BIM authoring environments such as Bonsai in Blender mainly expose functionality through graphical interfaces, where users select element types, adjust parameters, confirm actions, and place geometry in the 3D viewport. Such multi-step GUI workflows are not directly suitable for LLMs, which operate on text and structured inputs rather than interactive visual actions. Our approach maps these workflows into single tool calls with explicit parameters. For example, wall creation in Bonsai involves several GUI steps but is represented in the framework by one operation with arguments. Figure 3 illustrates this mapping. More generally, tool interfaces are designed around the strengths of LLMs rather than exactly mirroring the structure of the original GUI or low-level APIs. Following common best practices (Qin et al., 2023; Schick et al., 2023), our tools use clearly defined parameters, perform single operations, and return structured outputs which improves reliability in multi-step workflows.

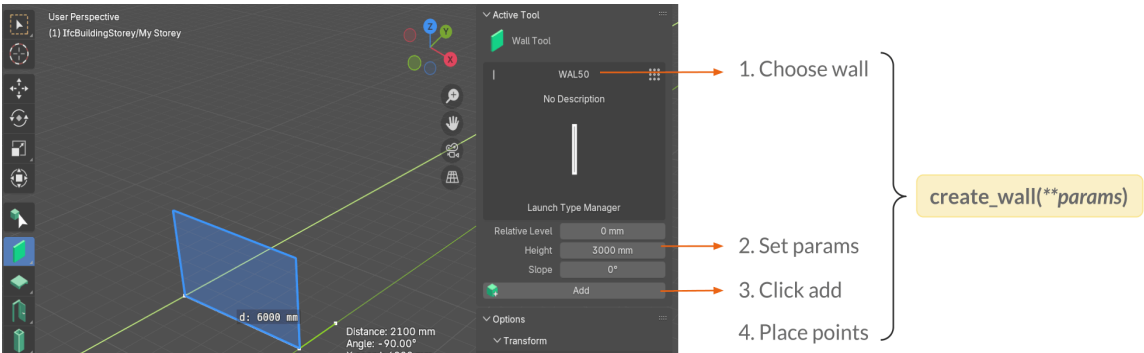


Figure 3: Mapping GUI operations to LLM tools. Creating a wall in the Bonsai add-on requires multiple GUI steps and is encapsulated into a single tool call.

4. Experiments and Results

We evaluate the framework through three experiments. The first experiment uses a custom IFC benchmark to compare the full toolset pipeline with configurations that use only the code generation for measuring the contribution of the predefined tools. The second experiment evaluates the framework on the existing IFC-bench-v1 dataset for scene querying to show how well the framework generalizes across different LLMs. The third experiment demonstrates practical geometry generation and multi-turn iterative design workflows using Claude Desktop as the LLM client.

4.1 Custom IFC Benchmark

To evaluate the contribution of our tool framework, we constructed a custom benchmark of 100 tasks covering retrieval (44), creation (30), and editing (26) operations across three difficulty levels: simple (44), medium (15), and complex (41). We use three IFC files of varying complexity (BasicHouse.ifc, TallBuilding.ifc, and LargeBuilding.ifc) from Wisén (2020). Tasks range from counting elements and reading attributes to multi-element construction (e.g., “create a room with four walls, a floor slab, a door, and a window”) and multi-step spatial reasoning (e.g., “how many doors to pass through to go from this bed to that bed”). Each task is scored automatically using a Python test script that uses the IfcTester library⁴ from the IfcOpenShell toolkit (1.0 correct, 0.5 partial, 0.0 incorrect).

We compare three configurations using Claude Sonnet 4.5 and GPT 5.4. Code-Once generates only a single code block without execution feedback, testing single-attempt code generation. Code-Exec provides iterative access to the `execute_ifc_code` tool with error feedback for up to 20 iterations, testing whether code generation alone with feedback is sufficient. All Tools provides access to all 52 tools from Section 3, allowing the LLM to freely choose between predefined tools and code generation. The IFC file is pre-loaded into Blender through the MCP server before each task, and an automated script launches Blender with the MCP server to run tasks without manual intervention. Table 2 shows the results. With Sonnet 4.5, both iterative configurations (Code-Exec and All Tools) achieve 86.0% overall accuracy, improving substantially over Code-Once at 53.2%. With GPT 5.4, Code-Exec reaches 82.9% and All Tools 82.4%, both above the 73.7% of Code-Once. This confirms across both models that execution feedback is essential for reliable IFC operations.

Table 2: Custom IFC benchmark results by difficulty for Claude Sonnet 4.5 and GPT 5.4. Output tokens reflect the total generated across all 100 tasks.

Model	Difficulty	N	Code-Once	Code-Exec	All Tools
Sonnet 4.5	Simple	44	70.5%	95.5%	88.6%
	Medium	15	20.0%	80.0%	86.7%
	Complex	41	39.0%	73.2%	80.5%
	Overall	100	53.2%	86.0%	86.0%
	Output tokens		66K	262K	114K
GPT 5.4	Simple	44	79.6%	93.2%	88.6%
	Medium	15	86.7%	80.0%	80.0%
	Complex	41	58.5%	68.3%	70.7%
	Overall	100	73.7%	82.9%	82.4%
	Output Tokens		70K	89K	34K

⁴ <https://docs.ifcopenshell.org/ifctester.html>

A consistent finding across both models is that All Tools generates substantially fewer output tokens than Code-Exec: $2.3\times$ fewer with Sonnet 4.5 (114K vs. 262K) and $2.6\times$ fewer with GPT 5.4 (34K vs. 89K). Predefined tools replace verbose code generation with compact tool calls, reducing both cost and latency since output tokens are typically $3\text{-}5\times$ more expensive than input tokens. This benefit holds across both models, confirming that tool-based interaction provides a general mechanism for reducing output token usage. The tradeoff is that all tool descriptions require around 40K input tokens in the system prompt. This cost can be reduced with prompt caching and on-demand tool loading (Shi et al., 2025), where only the tools needed for a given task are included.

4.2 Scene Querying Evaluation on IFC-bench-v1

In this experiment, we focus on scene querying using the existing IFC-bench-v1 benchmark (Hellin et al., 2025) and compare three models: GPT-5.4, Claude Sonnet 4.5, and GPT-5 mini. The same set of 52 tools is available to all models, and we evaluate their performance on the 103 questions without additional fine-tuning or prompt engineering. Following Hellin et al. (2025), the questions are grouped into three categories: direct tasks (44), which require extracting information directly from the IFC model; indirect tasks (30), which require aggregation or computation such as counting elements or calculating areas; and insufficient tasks (29), where the required information is not present in the IFC model and the model must recognize that the query cannot be answered from the data. This experiment helps us understand how well the framework generalizes across different LLMs and which types of questions remain challenging for current models. We conduct the experiment with GPT-5.4, Claude Sonnet 4.5, and GPT-5 mini, and manually check the correctness of the answers. As shown in Table 3, GPT-5.4 achieves the highest accuracy of 73.33%, followed by Sonnet 4.5 at 71.8% and GPT-5 mini at 65.0%. All models perform well on tasks with directly available information (70-82%), and they also perform well on insufficient-information tasks (86-97%). This shows that current LLMs can reliably determine whether information is present in the IFC model and retrieve it when available. As expected, indirect tasks remain the most difficult because they require the model to retrieve multiple values and combine them through additional reasoning steps.

Analyzing tool usage for correctly solved tasks with Sonnet 4.5, we find that the code execution tool accounts for 54.8% of all invocations, followed by scene overview and object query tools at 21.6% and 11.8%, respectively. This is expected because aggregation tasks such as counting, filtering by type, and reasoning over relationships require procedural logic.

Table 3: Accuracy on IFC-bench-v1 (Hellin et al., 2025) for three models using the full toolset.

Category	GPT-5.4			Sonnet 4.5			GPT-5 mini		
	Corr	Incorr	Acc	Corr	Incorr	Acc	Corr	Incorr	Acc
Direct	36	8	81.82%	34	10	77.3%	31	13	70.5%
Indirect	13	19	40.62%	12	18	40.0%	11	19	36.7%
Insufficient	28	1	96.55%	28	1	96.6%	25	4	86.0%
Total	77	28	73.33%	74	29	71.8%	67	36	65.0%

4.3 IFC Generation and Interactive Workflows

Beyond structured benchmarks, we demonstrate the framework’s capability for IFC generation by connecting our framework to Claude Desktop, an interactive LLM client.

Figure 4 shows IFC geometry generated from six single prompt tasks and compares the full toolset (top row) with Code-Exec only (bottom row). The predefined creation tools produce more coherent structures, especially for standard building elements because the tools encode correct IFC patterns. For non-standard geometry such as bridges or curved structures, the model combines predefined tools with custom mesh generation. IFC Code generation alone often produces disconnected or incomplete geometry on complex prompts. Despite the visual coherence, the generated IFC files from both configurations still lack some production BIM features such as complete property sets, correct space definitions, and material layer assignments. These examples show the current capabilities of the framework and the LLMs, and they also show that both still need improvement. We believe the tool-based approach provides a promising direction because it allows more domain knowledge and constraints to be encoded into the tools, which can guide the LLM toward generating higher-quality BIM models.

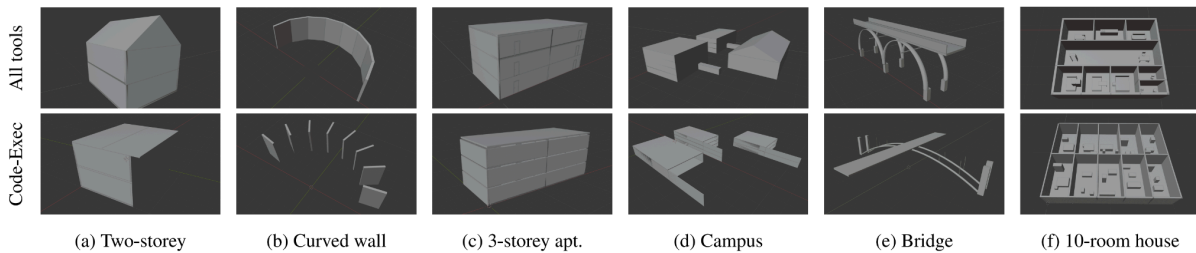


Figure 4: IFC geometry generation from text prompts with Claude Sonnet 4.5. Top row: All tools. Bottom row: Code-Exec only. The prompts are shortened here for display.

We also test multi-turn generation with all tools enabled. Figure 5 shows a six-step workflow where the model incrementally builds an L-shaped house with a floor slab, perimeter walls, door insertion, second-floor slab, hip roof, and styling. Each step builds on the previous geometry, and the model correctly references existing elements when adding new ones. This capability extends to any tool-compatible LLM system, and demonstrates that the framework can serve as a general interoperability layer between LLMs and BIM software.

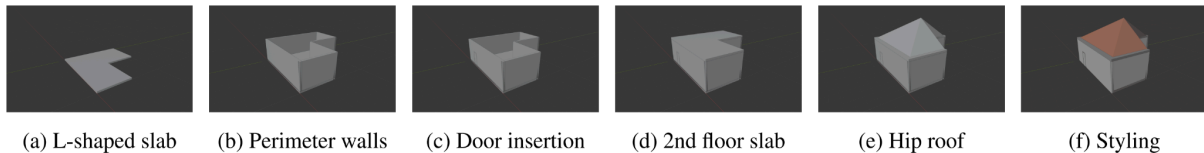


Figure 5: Multi-turn generation of an L-shaped building with Claude Sonnet 4.5 using all tools.

5. Discussion

The key insight from our experiments is that iterative execution feedback is more important than any single toolset. Allowing the model to execute actions, observe errors, and revise decisions makes IFC interaction much more reliable than one-shot generation. Predefined tools and dynamic code execution are complementary. Predefined tools improve efficiency and consistency for common structured operations, while code execution remains necessary for procedural and non-standard tasks. These results also show that improving tools alone is not sufficient. As the number of available tools increases, performance can decline if the underlying LLM cannot reliably select and use them (Shi et al., 2025). Retrieval support can further improve performance, especially for smaller models by grounding code generation in

relevant API documentation and usage examples (Patil et al., 2023). Future progress in tool-based BIM systems therefore depends on both better tool design and stronger LLM reasoning, planning, and code generation capabilities. Overall, the combined approach provides a practical path forward for robust BIM interaction.

6. Conclusion and Future Work

We presented an open-source, tool-based framework that enables LLMs to query, edit, and generate IFC-based BIM data through a structured set of 52 tools. By operating directly on the IFC standard through IfcOpenShell, the framework provides an open and platform-independent interface for AI-assisted BIM workflows. The architecture combines predefined parametric tools, dynamic code generation, and retrieval support. Our experiments show that iterative execution feedback improves accuracy, and that predefined tools help reduce output tokens compared to code generation alone. On IFC-bench-v1, the framework paired with GPT-5.4 achieves up to 73.3% accuracy without task-specific toolbox engineering. IFC generation and multi-turn design further demonstrate that the framework supports interactive BIM workflows.

Several directions remain for future work. Stronger BIM benchmarks that include IFC schema validation and semantic completeness checks are needed for production-level evaluation. Fine-tuning models on IFC-specific tasks could improve generative accuracy, especially for complex geometry and spatial reasoning where current models still struggle. Higher-level design abstractions such as room-level operations or building templates could reduce the number of tool calls required for complex designs. User studies with BIM practitioners are needed to evaluate the usability of natural-language BIM interaction in practice. Finally, connecting the framework to downstream construction applications such as compliance checking, quantity take-off, and digital twin workflows would help position it as a general interoperability layer for AI-assisted open BIM workflows.

References

- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). Language models are few-shot learners. *Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS '20*.
- Dawson-Haggerty, M., Bacher, J., Brown-Bivins, A., Forest, C., & O'Kane, J. (2019). *Trimesh: Load and analyze triangular meshes (version 3.2.0) [computer software]*. <https://trimesh.org/>.
- Deng, Z., Du, C., Nousias, S., & Borrmann, A. (2025). BIMgent: Towards autonomous building modeling via computer-use agents. *arXiv Preprint arXiv:2506.07217*.
- Dong, Y., Zhan, Z., Hu, Y., Doe, D. M., & Han, Z. (2025). AI BIM coordinator for non-expert interaction in building design using LLM-driven multi-agent systems. *Automation in Construction*, 180, 106563.
- Du, C., Esser, S., Nousias, S., & Borrmann, A. (2024). Text2BIM: Generating building models using a large language model-based multi-agent framework. *arXiv Preprint arXiv:2408.08054*.
- Du, C., Nousias, S., & Borrmann, A. (2024). Towards a copilot in BIM authoring tool using a large language model-based agent for intelligent human-machine interaction. *arXiv Preprint arXiv:2406.16903*.
- Duggempudi, J., Gao, L., Senouci, A., Han, Z., & Zhang, Y. (2025). Text-to-layout: A generative workflow for drafting architectural floor plans using LLMs. *arXiv Preprint arXiv:2509.00543*.
- Fernandes, D., Garg, S., Nikkel, M., & Guven, G. (2024). A GPT-powered assistant for real-time interaction with building information models. *Buildings*, 14(8), 2499.

- Guo, P., Xue, H., Ma, J., & Cheng, J. C. (2025). Archim: 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*.
- Hellin, S., Nousias, S., & Borrmann, A. (2025). Natural language information retrieval from BIM models: An LLM-based agentic workflow approach. *2025 European Conference on Computing in Construction (CIB W78 Conference on IT in Construction)*.
- ISO 16739-1. (2024). *Industry foundation classes (IFC) for data sharing in the construction and facility management industries — part 1: Data schema*. International Organization for Standardization. <https://www.iso.org/standard/84123.html>
- Jang, S., Lee, G., Oh, J., Lee, J., & Koo, B. (2024). Automated detailing of exterior walls using NADIA: Natural-language-based architectural detailing through interaction with AI. *Advanced Engineering Informatics*, 61, 102532.
- Jia, B., Chen, Y., Yu, H., Wang, Y., Niu, X., Liu, T., Li, Q., & Huang, S. (2024). *SceneVerse: Scaling 3D vision-language learning for grounded scene understanding*. <https://arxiv.org/abs/2401.09340>
- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS '20*.
- Li, A., Wong, P. K.-Y., Tao, X., Ma, J., & Cheng, J. C. (2025). An interactive system for 3D spatial relationship query by integrating tree-based element indexing and LLM-based agent. *Advanced Engineering Informatics*, 66, 103375.
- Li, R., Li, S., Kong, L., Yang, X., & Liang, J. (2025). *Zero-shot 3D visual grounding from vision-language models*. <https://arxiv.org/abs/2505.22429>
- Liu, B., & Chen, H. (2025). BIMCoder: A comprehensive large language model fusion framework for natural language-based BIM information retrieval. *Applied Sciences*, 15(14), 7647.
- Longo, A., Chung, C., Palieri, M., Kim, S.-K., Agha, A., Guaragnella, C., & Khattak, S. (2025). *Pixels-to-graph: Real-time integration of building information models and scene graphs for semantic-geometric human-robot understanding*. <https://arxiv.org/abs/2506.22593>
- Patil, S. G., Zhang, T., Wang, X., & Gonzalez, J. E. (2023). Gorilla: Large language model connected with massive APIs. *ArXiv, abs/2305.15334*. <https://api.semanticscholar.org/CorpusID:258865184>
- Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., Cong, X., Tang, X., Qian, B., et al. (2023). Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv Preprint arXiv:2307.16789*.
- Schick, T., Dwivedi-Yu, J., Dessí, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. *Proceedings of the 37th International Conference on Neural Information Processing Systems, NIPS '23*.
- Succar, B. (2009). Building information modelling framework: A research and delivery foundation for industry stakeholders. *Automation in Construction*, 18(3), 357–375. <https://doi.org/https://doi.org/10.1016/j.autcon.2008.10.003>
- Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., & Anandkumar, A. (2023). Voyager: An open-ended embodied agent with large language models. *arXiv Preprint arXiv:2305.16291*.
- Wisén, A. (2020). *Bim-whale-ifc-samples: IFC sample files for the BIM whale project*. GitHub repository. <https://github.com/andrewisen/bim-whale-ifc-samples/tree/main>
- Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). React: Synergizing reasoning and acting in language models. *International Conference on Learning Representations (ICLR)*.
- Zheng, J., & Fischer, M. (2023). Dynamic prompt-based virtual assistant framework for BIM information search. *Automation in Construction*, 155, 105067.
- Shi, Z., Wang, Y., Yan, L., Ren, P., Wang, S., Yin, D. and Ren, Z., 2025, July. Retrieval models aren't tool-savvy: Benchmarking tool retrieval for large language models. In Findings of the Association for Computational Linguistics: ACL 2025 (pp. 24497-24524).