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

Bharathi Kannan Nithyanantham[1*] (✉), Tobias Sesterhenn[2*], Ashwin Nedungadi[1], Sergio Peral Garijo[2], Janis Zenkner[2], Christian Bartelt[2], and Stefan Lüdtke[1]

[1] University of Rostock, Rostock, Germany
[2] Clausthal University of Technology, Clausthal-Zellerfeld, Germany
bharathikannan.nithyanantham@uni-rostock.de

Introduction Building Information Modeling (BIM) provides a shared digital representation of buildings and supports collaboration throughout the AEC lifecycle. The Industry Foundation Classes (IFC) standard enables open and vendor-independent exchange of BIM information. Despite this standardization, creating and modifying BIM models remains a complex task requiring expert knowledge of authoring tools and data structures. Recent advances in Large Language Models (LLMs) demonstrate their ability to interpret natural language instructions, perform reasoning, and interact with external tools. This creates the possibility of AI assistants that translate high-level design intentions into BIM operations. Current approaches, however, are frequently coupled to proprietary APIs such as Revit or Vectorworks, reducing interoperability. To address this limitation, we introduce IFC-Copilot, a framework that allows LLM agents to operate directly on IFC data. The system provides a standardized interface for querying, editing, and generating BIM models while remaining independent of commercial BIM platforms.

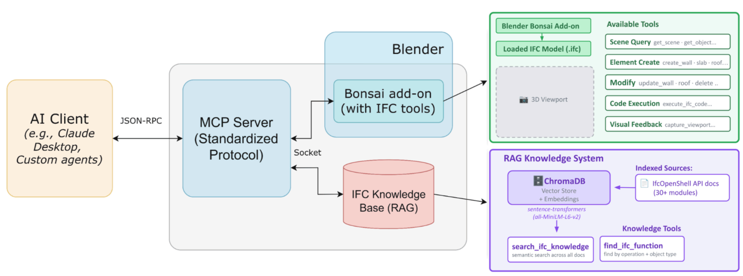

Fig. 1: System Architecture

Methodology The proposed framework follows a tool-based agent architecture composed of an LLM client, a Model Context Protocol (MCP) server, and a Blender extension using IfcOpenShell for IFC manipulation. At initialization, available BIM operations are provided to the LLM as JSON-based tool descriptions. During interaction, the model selects an appropriate tool, generates the required parameters, receives execution feedback, and iteratively continues until the requested task is completed. The framework contains three complementary tool layers. The first layer consists of **predefined IFC tools**, which provide parametric implementations for common BIM operations such as scene querying, the creation of walls, slabs, roofs, doors, and windows, element modification, material handling, and visual feedback. The second layer provides **dynamic code execution**, enabling the LLM to generate Python scripts based on IfcOpenShell for tasks that exceed the predefined toolset. These scripts are executed in a restricted environment that returns feedback for iterative error correction and refinement. The third layer integrates **retrieval-augmented generation**, where IFC documentation and API information are indexed in a vector database and retrieved during code generation to improve correctness and maintain compatibility with evolving APIs.

Table 1: Custom benchmark results by difficulty for Claude Sonnet 4.5 and GPT 5.4.

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

Evaluation The framework is evaluated using three experiments. First, a custom benchmark containing 100 IFC tasks is created, covering information retrieval, model creation, and editing. Three configurations are compared: one-shot code generation, iterative code execution, and the complete tool framework. As shown in Table 1 the execution feedback is essential for reliable BIM interaction. Claude Sonnet 4.5 improves from 53.2% accuracy with single-step code generation to 86.0% using iterative execution. GPT 5.4 improves from 73.7% to above 82%. The complete tool configuration achieves similar accuracy while substantially reducing generated output tokens. Compared with pure iterative code generation, predefined tools reduce output tokens by approximately 2.3–2.6$\times$ because complex code blocks are replaced by compact structured tool calls. Second, we evaluate the system on the IFC-bench-v1 querying benchmark. The framework achieves up to 73.3% accuracy with GPT 5.4 without benchmark-specific optimization. Direct information retrieval and detection of unavailable information are solved reliably, while complex indirect reasoning tasks remain challenging. Finally, interactive generation experiments demonstrate multi-turn BIM workflows. The LLM incrementally creates buildings by adding slabs, walls, doors, roofs, and styles while maintaining references to existing IFC elements.

Discussion & Future Work Our experiments show that iterative execution feedback is the main factor enabling reliable LLM interaction with IFC models. Predefined tools and dynamic code generation provide complementary advantages: tools improve efficiency and enforce correct BIM patterns, while code generation supports uncommon operations and complex procedural tasks. Remaining challenges include complex spatial reasoning, semantic completeness of generated IFC models, and automatic selection among large tool collections. Future work will therefore investigate stronger BIM benchmarks, IFC-specific model adaptation, higher-level design tools, and integration with downstream construction workflows.