Towards Agent: LLM-based Framework for Robotic Construction by Leveraging IFC

Song Du[1,†], Yan Gao[1,†], Wei Tong, and Yiwei Weng *

[1]Department of Building and Real Estate, The Hong Kong Polytechnic University, Hong Kong
*Corresponding author yiwei.weng@polyu.edu.hk
[†]These authors contributed equally to this work

Abstract

This work explores the integration of Large Language Model (LLM)-powered agents into robotic construction workflows. An LLM-based agent framework by leveraging standardized IFC format is proposed to guide robotic tasks, such as bricklaying and 3D concrete printing. Through a combination of tailored tools, prompt templates, and a *CustomMemory* module, the framework enables efficient extraction of building information and execution of robotic tasks. Finally, experimental validation in a simulated environment demonstrates the framework's potential to streamline robotic construction workflows from textual or voice commands to physical construction activities, while reducing redundant operations and enhancing reliability through thoughtful design.

Keywords – LLM Agents, Robotic Construction, IFC, BIM, 3D Concrete Printing

1 Introduction

The construction industry is witnessing rapid advancements in automation, driven by the need to address challenges such as large-scale tasks, stringent precision requirements, and dynamic environments. Robotics has emerged as a promising solution, enabling efficient and precise execution of tasks like bricklaying and 3D concrete printing. Despite significant progress in robotic construction, the integration of large language models (LLMs) into these workflows remains limited, particularly for tasks requiring seamless communication between building information modeling (BIM) data and robotic operations. While LLMs have shown great potential in robotics operations through their emergent and reasoning capabilities, their applications have mainly focused on the design stage, where generative tools powered by LLMs are widely adopted. However, their use in robotic construction processes remains largely unexplored.

This work addresses this gap by proposing an IFC-enhanced LLM agent framework for robotic construction. The framework leverages the standardized Industry Foundation Classes (IFC) format to extract building information efficiently and accurately. Furthermore, by incorporating tailored tools, structured prompt templates, and a designed *CustomMemory* module, the framework effectively guides robots in performing tasks such as bricklaying and 3D concrete printing. Experimental validation in a simulated environment demonstrates the framework's effectiveness in streamlining workflows, avoiding redundant operations, and enhancing reliability.

2 Related Works

2.1 Robotic Construction and LLM

LLMs have shown significant potential in robotics operations, enabling tasks such as natural language and code generation, numerical prediction, motion planning, and fault detection [1]. Works like RoboCodeX [2] and Reasoning Tuning [3] integrate visual and physical data in multimodal models to generate executable numeric predictions, achieving cross-platform and cross-scenario generalization by decomposing high-level semantics into traceable low-level actions. Methods like ReKep [4] enhance skill planning by identifying critical points, allowing robots to handle complex environments effectively, while Code-as-Monitor [5] combines Vision Language Models (VLMs) with real-time monitoring to predict risks and detect errors proactively.

Robots are advancing automation in construction, particularly in bricklaying and 3D concrete printing. For instance, mobile robots are used for automated bricklaying, integrated within BIM [6]. 3D concrete printing allows robotic arms to fabricate components efficiently, reducing material waste [7]. However, challenges in efficiency and structural stability persist, especially in large-scale projects. While robotic construction progresses, integration with LLMs remains limited. In contrast, LLM-driven tools like text2BIM [8] and AIstructure-Copilot [9] are making strides in generating layouts and BIM models from textual

descriptions during the design phase.

2.2 IFC Standardization

The IFC was introduced by buildingSMART International, to address interoperability challenges within the Architecture, Engineering, and Construction (AEC) and facility management (FM) industries [10]. IFC provides a standardized, neutral, and non-proprietary data format that spans the entire lifecycle of construction projects [11], including design, construction, and maintenance. By promoting seamless communication and reducing reliance on proprietary formats, IFC has become a universal standard in domains such as buildings, bridges, airports, and railways [12]. Leading AEC software platforms, including Autodesk and Bentley Systems, have adopted IFC to facilitate interoperability and enable efficient data exchange across stakeholders.

IFC's structured representation of building elements and attributes, accessible via tools like IfcOpenShell [13], supports automated construction workflows. By extracting dimensions, material properties, and placement information from IFC elements such as IfcWall and IfcBeam [14], the format facilitates integration with downstream processes like robotic assembly. Technologies like ifcJSON [15] further enhance this integration by converting IFC data into structured formats compatible with modern workflows. Leveraging IFC data in robotic construction, combined with LLM, has significant potential to address key industry challenges like large-scale task automation and precision requirements. For instance, LLM agents can potentially guide robots to efficiently and accurately perform complex tasks based on IFC files, including wall masonry and rebar tying.

3 Methodology

This work builds on prior research in the Text2BIM framework [8], [16], which utilizes LLM-based multi-agent systems to convert natural language commands into editable 3D building models. Extending this concept to robotic construction, our approach transforms engineers' natural language instructions into actionable robotic tasks based on BIM. A cornerstone of the approach is the use of IFC, offering a standardized and structured format for building design data to enable accurate extraction and utilization.

Directly processing IFC data with LLMs poses challenges due to its complexity and domain specificity, often resulting in parsing inaccuracies. Moreover, token limitations further restrict handling large files. To address these issues, IfcOpenShell [13] is employed for precise data extraction, effectively avoiding token limitations associated with processing large IFC files and ensuring efficient and structured retrieval of BIM data.

Hence, a suite of tools based on IfcOpenShell is developed for LLM-based agents to handle necessary calculations and generate operational code in robotic construction. The workflow, as shown in Figure 1, demonstrates seamless integration of tool usage, memory modules, and task performance to transform BIM models into robotic construction operation.

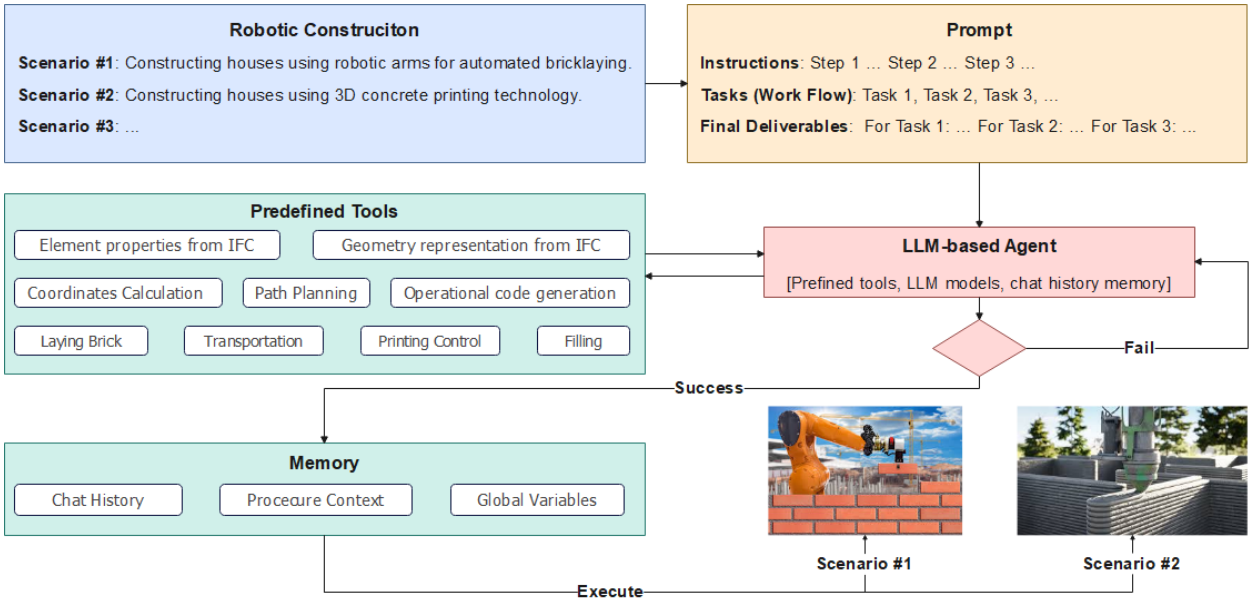

Figure 1. Proposed LLM agent-based Robotic Construction Framework

3.1 Toolset Design

Tools are designed from three key perspectives: (a) information extraction from IFC files, including both entity objects and geometric representations. Entity objects include building components such as walls, doors, windows, beams, columns, and roofs, while geometric representations cover attributes like shapes, placement, profiles, dimensions, extrusions, orientations, and spatial relationships. (b) essential calculations, such as brick coordinate calculation and printing path planning. (c) robotic operations, like bricklaying, printing control, material transportation, and filling.

Table 1 Examples of predefined tools

Name	Description	Outcome
LoadIFC File	Load an IFC file and return the ifcopenshell.file object.	The IFC file has been successfully loaded.
Extract Wall	Extract wall properties from an IFC file using ifcopenshell.	Wall properties are stored as variable *all_wall_props_global*
Calculate Bricks	Calculate brick coordinates based on wall dimensions and brick size. Input is a JSON format with keys: wall_properties, door_properties, window_properties, and brick_size.	Filtered brick coordinates are stored as variable filtered_brick_coordinates_global with {len(filtered_brick_coordinates_global)} records.
LayBricks	Lay bricks at the specified coordinates. Input is JSON format containing keys: filtered_brick_coordinates.	The system has successfully laid bricks at the specified coordinates.

The context of each tool is structured as three key components: 1) Name, to identify the tool during operation, 2) Description, to guide invocation and input format, and 3) Outcome, detailing the output variables and their storage locations, which are explicitly incorporated into memory module for LLM's contextual awareness. Table 1 presents examples of predefined tools and their descriptions. Prompt Template

The prompt template used in this work consists of three key components: Instructions, Tasks, and Final deliverables.

Instructions include directives to ensure: a) check for required data before each task; b) reuse available data from memory; c) use the provided tools and update results; d) transit between tasks with clear statements; 5) provide reasoning and results for each task before proceeding.

Task prompts are designed in two styles: a) open-ended prompt: providing a high-level instruction that allows the LLM to determine the appropriate tools and workflow based on its emergent and reasoning capabilities; b) tool-specific prompt: combining tool descriptions with input variables to guide precise tool invocation and task execution.

Final deliverables specify the expected output, which is stored in the memory module's chat history to ensure consistent progress tracking and provide context for subsequent tasks. Additionally, tools can be re-invoked at this stage to perform supplementary functions, such as saving output in a specific file format.

The tailored prompt template is shown in Figure 2.

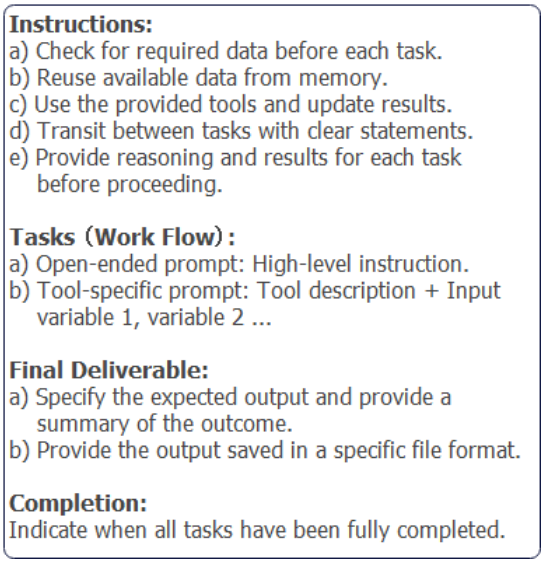

Figure 2. Structured prompt template

3.2 Memory Module

The memory module in this work consists of three components: 1) chat history, 2) output variables, and 3) procedure context. Using *ConversationBufferMemory* in LangChain [17], each prompt and the corresponding answers from the final deliverables section are recorded in the chat history. Meanwhile, the output variables during tool invocation are explicitly stored (e.g., as

global variables) and returned in an outcome summary, informing the agent to reuse these variables within the same session. However, once a session is completed, the outcomes generated during the process are discarded, leading to recalculations of previously generated variables in subsequent prompts.

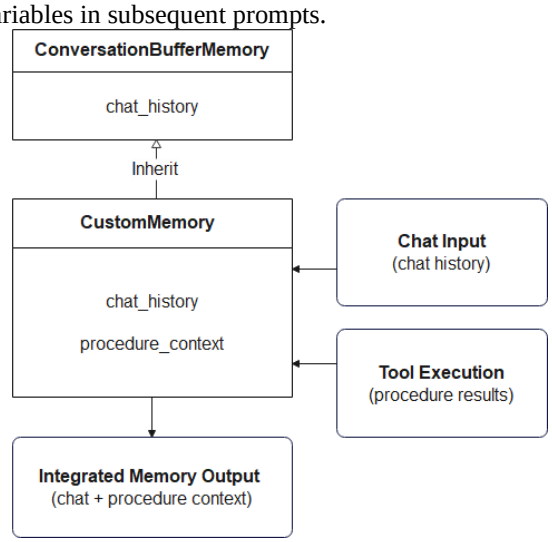

Figure 3. *CustomMemory* class and data flow

To address this issue, a *CustomMemory* class is designed, as shown in Figure 3, extending from *ConversationBufferMemory*, to retain chat history and store tool execution outcomes (i.e., procedure context). This integration enables the agent to review procedure results in any session, ensuring informed decisions and avoiding redundancy.

However, it is worth noting that memory content may continuously accumulate over multiple sessions, potentially leading to the scalability challenge. To mitigate this, efficient context management strategies, such as hierarchical summarization, may be required in future development to optimize memory usage while preserving relevant procedural data.

4 Proof of Concept

4.1 Preparation

To validate the effectiveness of the proposed LLM-based autonomous agent framework in robotic construction, experiments were conducted for two different scenarios: robotic bricklaying and 3D concrete printing. The experimental setup featured a simple structure comprising four walls, a door, and two windows, as shown in Figure 4. For simplicity, the bricks used in wall construction were cubic, with dimensions matching the wall thickness. Likewise, the layer height for 3D concrete printing was set equal to the wall thickness.

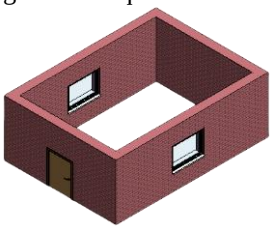

Figure 4. Experimental Structure

LangChain is adopted as the development foundation and a series of tools are developed for the experiment. The GPT-4 is utilized as the LLM engine, with the prompts shown in Figure 5.

Instructions:
0. Complete all tasks sequentially in the given order. Do not stop until all tasks are fully completed.
1. Before performing each task, explicitly use the QueryMemory tool to determine if the required information is already available, even if it is the first task.
2. If any part of the required information is already available in memory, reuse it instead of recalculating or re-extracting it.
3. Use the provided tools to complete the remaining parts of the tasks, storing results in the indicated global variables. Confirm that the variables are updated for subsequent use.
4. After completing each task, explicitly transition to the next task by stating, "Task [N] completed. Moving to Task [N+1]."
5. Provide clear reasoning and results for each task before transitioning to the next.

Tasks:
1. Load the project3.ifc file.
2. Extract all the elements from the loaded ifc object.
3. Calculate brick coordinates for the walls based on wall dimensions and brick size (300 mm cubic bricks), then filter out coordinates conflict with other structural elements or openings.
4. Generate 3D concrete printing paths for walls based on wall dimensions and layer height (300 mm), avoiding areas occupied by other structural elements or openings.
5. Lay bricks at the specified coordinates provided in the `filtered_brick_coordinates` input. Ensure all coordinates are processed, and return the result, including the number of bricks laid.
6. Print concrete along the 3D paths provided in the `layers_print_paths` input. Ensure all paths are processed, and return the result, including the number of layers printed.

Final Deliverable:
For Task 3: Save the filtered brick coordinates into a JSON file named 'filtered_brick_coordinates.josn'.
For Task 4: Save the 3D concrete printing paths into a JSON file named 'printing_paths.json'.
For Task 5: Summarize the brick-laying task, including the total number of bricks laid.
For Task 6: Summarize the 3D concrete printing task, including the total number of layers printed.

Completion:
Indicate when all tasks have been fully completed.

Figure 5. Prompts in the experiment

```
The first task is to read the project3.ifc file and initialize processing. However, before I do that, I need to check if the file has
already been loaded into memory.
Action: QueryMemory
Action Input: {}
Observation: Memory is empty. No prior actions or outputs are available.
Thought:Since the memory is empty, the project3.ifc file has not been loaded yet. I will now load the file using the LoadIFCFile tool.
Action: LoadIFCFile
Action Input: {"file_path": "project3.ifc"}{'file_path': 'project3.ifc'}
project3.ifc
<ifcopenshell.file.file object at 0x000001E5235BC610>

Observation: project3.ifc loaded successfully and stored as variable ifc_object.
Thought:The project3.ifc file has been successfully loaded and stored in the variable ifc_object. Task 1 is now completed. Moving to
Task 2, which is to identify and extract architectural elements from the IFC file.
Action: ExtractWallProperties
Action Input: {"ifc_object": "ifc_object"}
Observation: Wall properties stored as variable all_wall_props_global with 4 records.
Thought:The wall properties have been successfully extracted from the IFC file and stored in the variable all_wall_props_global. Now, I
will extract the door properties.
Action: ExtractDoorProperties
Action Input: {"ifc_object": "ifc_object"}
Observation: Door properties are stored as variable all_door_props_global with 1 records.
Thought:The door properties have been successfully extracted from the IFC file and stored in the variable all_door_props_global. Now, I
will extract the window properties.
Action: ExtractWindowProperties
Action Input: {"ifc_object": "ifc_object"}
Observation: Window properties are stored as variable all_window_props_global with 2 records.
Thought:The window properties have been successfully extracted from the IFC file and stored in the variable all_window_props_global.
Task 2 is now completed. Moving to Task 3, which is to calculate and filter bricks for potential 3D printing.
Action: CalculateAndFilterBricks
Action Input: {"wall_properties": "all_wall_props_global", "door_properties": "all_door_props_global", "window_properties":
"all_window_props_global", "brick_size": 300}
Observation: Filtered brick coordinates stored as variable filtered_brick_coordinates_global with 394 records.
Thought:The brick arrangement has been computed successfully, taking into account the size of the bricks and eliminating any conflicts
with doors and windows. The filtered brick coordinates are now ready for 3D printing path generation.
Action: Generate3DBrickPrintPath
Action Input: {"filtered_brick_coordinates": "filtered_brick_coordinates_global", "layer_height": 300}
Observation: 3D printing paths stored as the global variable layers_print_paths_global for 7 layers.
Thought:The 3D printing paths have been successfully generated and stored in the variable layers_print_paths_global. The paths are
structured into 7 layers, each with a height of 300 mm.
Action: QueryMemory
Action Input: {}
Observation:
### Procedure Context (Tools) ###
Tool: load_ifc_tool - Result: project3.ifc loaded successfully and stored as variable ifc_object.
Tool: extract_wall_tool - Result: Wall properties stored as variable all_wall_props_global with 4 records.
Tool: extract_doors_tool - Result: Door properties are stored as variable all_door_props_global with 1 records.
Tool: extract_windows_tool - Result: Window properties are stored as variable all_window_props_global with 2 records.
Tool: calculate_and_filter_brick_tool - Result: Filtered brick coordinates stored as variable filtered_brick_coordinates_global with
394 records.
Tool: calculate_and_filter_brick_tool - Result: 3D printing paths stored as the global variable layers_print_paths_global for 7 layers.

Action: ExecuteBrickPlacement
Action Input: {"brick_coordinates": filtered_brick_coordinates_global}
Action: {"StartBrickSupply", "RobotArmStart", "ArmMove from [supply_point] to [0.24,-0.36,0.15]",
 …,
"ArmMove from [supply_point] to [0.21,0,1.95]", "StopBrickSupply", "RobotArmStop"}
Thought:The brick laying task was completed, and a total of 394 bricks were laid.
Action: Execute3DPrinting
Action Input: {"print_paths": layers_print_paths_global}
Action: {"RobotArmStart", "MoveArm to layer_1_start", "StartExtrudeConcrete", "FollowPath layer_1_path", "FollowPath layer_2_path",
"FollowPath layer_3_path", "FollowPath layer_4_path", "FollowPath layer_5_path", "FollowPath layer_6_path", "FollowPath layer_7_path",
"StopExtrudeConcrete", "RobotArmStop"}
Thought:The 3D printing task is completed, with a total of 7 layers printed.
Final Answer: All tasks have been fully completed.
```

Figure 6. Tasks execution instructed by LLM agent

4.2 Experiment and Results

In the prompts, a robotic construction workflow is manually decomposed into a series of tasks. The experimental results demonstrate that the designed LLM-based agent can sequentially complete these tasks as specified, ultimately achieving robotic bricklaying and 3D concrete printing in a simulated environment.

The tasks execution instructed by the developed LLM agent is shown in Figure 6.

The tasks 5 and 6 execution instructed by the developed LLM agent is shown in Figure 7.

The experiments demonstrate that the proposed LLM-based framework can accurately invoke predefined tools according to the prompts. Moreover, the agent can consistently achieve reliable variable transmission through well-structured tool descriptions. For example, the building element properties are organized into *Action Input:{"wall_properties":all_wall_props_global,"door_properties":all_door_props_global,"window_properties":all_window_props_global,"brick_size":300}* based on the *CalculateBricks* tool description — "Input is a JSON format

with keys: wall_properties, door_properties, window_properties, and brick_size". Additionally, the agent can establish logical connections between tasks through well-designed outcome summaries in the results.

4.3 Efficiency and Reliability

The efficiency of *CustomMemory* is also validated in the experiment. For instance, when wall properties have already been extracted previously, and a task to extract properties of all elements is required, the agent can correctly identify that only the remained properties need to be extracted — *"The wall properties have already been extracted and stored in the variable all_wall_props_global. Now, I need to extract the properties of the doors and windows"*, demonstrating the ability to prevent the agent from redundant operations.

Similarly, both brick coordinates calculation and 3D printing path generation require extracting properties of all building elements. To quantify the efficiency improvement, we conducted controlled experiments to compare execution time with and without *CustomMemory* for the above tasks in a scenario where these properties had already been extracted in previous sessions.

As shown in Table 2 and Figure 8, the results indicate that *CustomMemory* reduces the execution time by over 60%. This is because, through the tailored procedure memory, the agent can recognize that the necessary data had already been extracted and stored, thereby avoiding unnecessary operations. In practice, the efficiency improvement is directly proportional to the number of previously generated and stored variables.

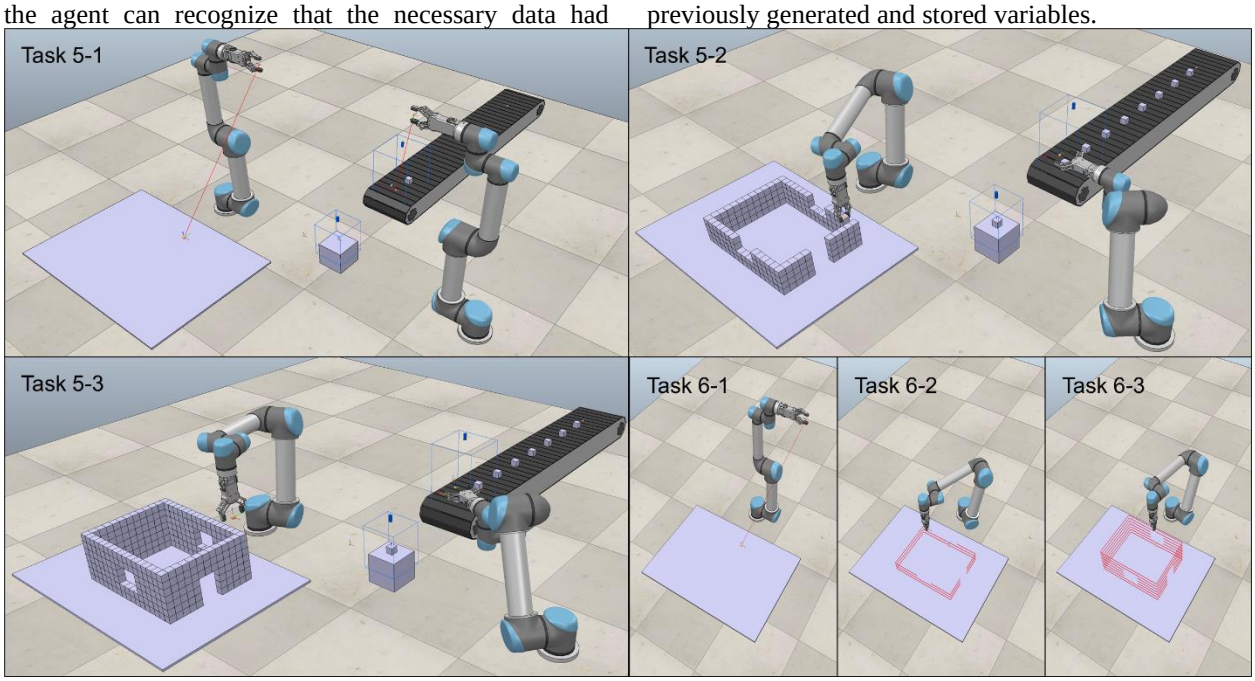

Figure 7. Tasks 5 and 6 executions instructed by LLM agent

Table 2 *CustomMemory* efficiency comparison

Tasks	Without CustomMemory	With CustomMemory
Brick Calculation	37.14s	13.67s
3D Path Generation	44.35s	15.87s

To assess the reliability of the proposed agentic framework, 10 similar prompts are generated using ChatGPT-4o to evaluate the execution success rate. Since the Instructions section is pre-designed as a fixed template, only the Tasks section is replaced in each prompt variation. Successful execution is measured based on how many tasks are correctly carried out.

The result shows that when the required parameters are explicitly specified (such as project3.ifc, 300-mm cubic brick, and 300-mm layer height), all 10 variations of the modified prompts successfully complete their respective tasks, demonstrating the excellent reliability of the proposed agentic framework and its robust adaptability to variations in input language.

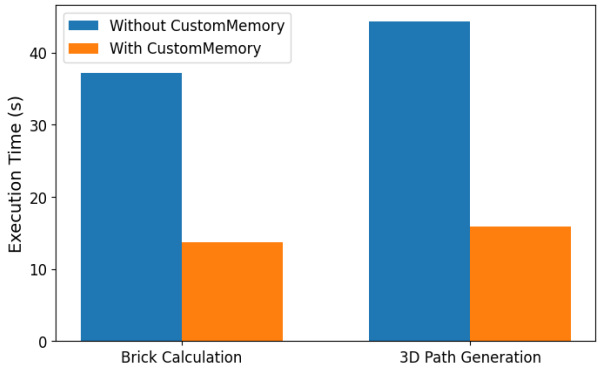

Figure 8. Execution time with vs without *CustomMemory*

However, it is worth noting that if these parameters are not explicitly provided (excluding variables generated dynamically by the agent during execution), the LLM tends to infer missing values, potentially leading to hallucinations. For example, it may assume a commonly used brick size, which could introduce unintended variations in execution.

5 Conclusion

This work presents an IFC-enhanced LLM agent framework for robotic construction, leveraging the standardized IFC format to enable efficient and accurate data extraction while avoiding computationally expensive approaches such as LLM fine-tuning or Retrieval-Augmented Generation (RAG). The tailored agent accurately invokes predefined tools through structured prompts, ensuring robust and reliable performance. Additionally, a *CustomMemory* module is developed to retain procedural context across sessions, significantly reducing execution time by eliminating redundant operations. The framework is validated in a simulation environment for robotic bricklaying and 3D concrete printing, demonstrating its effectiveness in automating construction workflows.

Furthermore, this work adopts an agentic workflow similar to Text2BIM, making it a natural extension of its approach. While Text2BIM primarily focuses on design-phase automation, this framework advances the concept by integrating LLM-driven robotic task execution, bridging the gap between BIM-based design generation and physical construction automation. Future research can explore the development of a full-process agentic pipeline, seamlessly connecting design-phase reasoning with robotic construction execution. This would enable an end-to-end intelligent workflow where LLMs autonomously generate, refine, and execute construction tasks based on BIM models, further enhancing efficiency, adaptability, customization, and automation in smart construction.

6 Discussion

Although the proposed framework demonstrates efficiency and reliability in simulation, transitioning to real-world robotic construction presents additional challenges. Integrating sensor feedback is crucial for real-time perception, as construction environments require adaptive decision-making based on LiDAR, cameras, and IMUs to handle unexpected obstacles. Error handling is essential for correcting deviations in robotic construction movements, such as bricklaying and 3D concrete printing, to ensure precise execution. Additionally, safety measures, including collision avoidance, emergency stop mechanisms, and regulation compliance, are required to ensure secure deployment in dynamic environments.

A promising solution to these challenges is the use of multiple agents to enhance coordination and execution. For example, instead of relying on a single agent, a multi-agent system could include: 1) a task planning agent, responsible for autonomous task decomposition and organization based on high-level construction goals; 2) an execution agent, handling low-level robotic control, ensuring precise movement, material placement, and process execution; 3) a supervisory agent, monitoring real-time sensor feedback, detecting errors, and triggering corrective actions when needed. By distributing responsibilities across specialized agents, this approach could improve reliability and scalability in real-world robotic construction.

Additionally, further investigation is needed to assess how the system handles ambiguous or conflicting instructions, particularly when multiple interpretations could lead to different robotic execution strategies. The risks of LLM hallucinations, such as generating incorrect or assumed parameters, require quantitative analysis to evaluate error rates and system robustness. Moreover, a deeper exploration of failure cases and error recovery mechanisms will be essential to ensuring the framework's adaptability and robustness in real-world robotic construction.

Furthermore, while the framework has already demonstrated some chain-of-thought (CoT) reasoning in its current stage. For example, executing Tasks 1, 5, and 6 in Figure 5 (skipping Tasks 2, 3, and 4), the agent can still extract building elements and their properties, as well as perform necessary calculations based on its reasoning and tool descriptions. In future work, we will continue to explore leveraging LLM's CoT capabilities to further enhance autonomous task decomposition and organization, improving the framework's adaptability in complex construction workflows.

References

[1] J. Wang *et al.*, 'Large Language Models for Robotics: Opportunities, Challenges, and Perspective', Jan. 2024, [Online]. Available: http://arxiv.org/abs/2401.04334

[2] Y. Mu *et al.*, 'RoboCodeX: Multimodal Code Generation for Robotic Behavior Synthesis', Feb. 2024, [Online]. Available: http://arxiv.org/abs/2402.16117

[3] J. Xu, S. Jin, Y. Lei, Y. Zhang, and L. Zhang, 'RT-Grasp: Reasoning Tuning Robotic Grasping via Multi-modal Large Language Model', Nov. 2024, [Online]. Available: http://arxiv.org/abs/2411.05212

[4] W. Huang, C. Wang, Y. Li, R. Zhang, and L. Fei-Fei, 'ReKep: Spatio-Temporal Reasoning of Relational Keypoint Constraints for Robotic Manipulation', Sep. 2024, [Online]. Available: http://arxiv.org/abs/2409.01652

[5] E. Zhou *et al.*, 'Code-as-Monitor: Constraint-aware Visual Programming for Reactive and Proactive Robotic Failure Detection', Dec. 2024, [Online]. Available: http://arxiv.org/abs/2412.04455

[6] B. Xiao, C. Chen, and X. Yin, 'Recent advancements of robotics in construction', Dec. 01, 2022, *Elsevier B.V.* doi: 10.1016/j.autcon.2022.104591.

[7] Y. Weng *et al.*, 'Comparative economic, environmental and productivity assessment of a concrete bathroom unit fabricated through 3D printing and a precast approach', *J Clean Prod*, vol. 261, Jul. 2020, doi: 10.1016/j.jclepro.2020.121245.

[8] C. Du, S. Esser, S. Nousias, and A. Borrmann, 'Text2BIM: Generating Building Models Using a Large Language Model-based Multi-Agent Framework', Aug. 2024, [Online]. Available: http://arxiv.org/abs/2408.08054

[9] S. Qin *et al.*, 'AIstructure-Copilot: Assistant for Generative AI-Driven Intelligent Design of Building Structures', *Smart Construction*, Mar. 2024, doi: 10.55092/sc20240001.

[10] 'buildingSMART International'. Accessed: Jan. 15, 2025. [Online]. Available: https://www.buildingsmart.org/

[11] S. T. Matarneh, M. Danso-Amoako, S. Al-Bizri, M. Gaterell, and R. Matarneh, 'Building information modeling for facilities management: A literature review and future research directions', Jul. 01, 2019, *Elsevier Ltd*. doi: 10.1016/j.jobe.2019.100755.

[12] 'Domains - buildingSMART International'. Accessed: Jan. 15, 2025. [Online]. Available: https://www.buildingsmart.org/standards/domains/

[13] 'IfcOpenShell - The open source IFC toolkit and geometry engine'. Accessed: Jan. 15, 2025. [Online]. Available: https://ifcopenshell.org/

[14] L. Zhang and N. M. El-Gohary, 'Automated IFC-based building information modelling and extraction for supporting value analysis of buildings', *International Journal of Construction Management*, vol. 20, no. 4, pp. 269–288, Jul. 2020, doi: 10.1080/15623599.2018.1484850.

[15] 'buildingsmart-community/ifcJSON: Repository containing the specification for IFC.JSON'. Accessed: Jan. 15, 2025. [Online]. Available: https://github.com/buildingsmart-community/ifcJSON

[16] C. Du, S. Nousias, and A. Borrmann, 'Towards a copilot in BIM authoring tool using a large language model-based agent for intelligent human-machine interaction'. [Online]. Available: https://huggingface.co/docs/transformers/transformers_agents

[17] 'LangChain'. Accessed: Jan. 15, 2025. [Online]. Available: https://www.langchain.com/