A Strongly Typed Assembly Agent for Construction Integrating Rhino and Multimodal Large Language Models

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

[1] Department of Building and Real Estate, The Hong Kong Polytechnic University, Hong Kong
*yiwei.weng@polyu.edu.hk

Abstract

Assembly tasks in constrction are characterized by customized components, complex connection relationships, and dynamic environments, making conventional approaches designed for fixed workflows difficult to reliably reuse. This paper proposes an assembly agent framework that integrates Rhino spatial data with multimodal large language models, enabling end-to-end closed-loop control from digital models to robotic execution. First, the framework encapsulates Rhino and robot functionalities as strongly typed APIs, constraining agent decisions within an executable action space to reduce hallucinated calls. Then, the framework establishes coordinate consistency between the Rhino space and the robot space, ensuring that planning and execution proceed under a unified reference frame. Thirdly, the framework employs a component color-annotation mechanism, that continuously reflects assembly progress in visual inputs while forming traceable evidence together with execution logs. Experiments conducted on two scaled assembly cases involving a table and a building façade demonstrate that both assembly sequence accuracy and safe planning rate exceed 95%, indicating that the proposed framework possesses sequence planning consistency and execution safety assurance capability under closed-loop decision-making.

Keywords

Assembly automation; Robotics; Agent

1 Introduction

Construction assembly automation is transitioning from mechanized execution of isolated processes to flexible assembly involving multiple components and steps [1]. On-site assembly tasks are typically customized, such as the positioning and installation of irregular components [2]. Given that component geometries vary substantially, task specifications change across projects. Additionally, site conditions are dynamic and complex, and therefore, robots are required to continuously perform component recognition, grasping, transportation, alignment, assembly, and fastening, while simultaneously complying with assembly quality requirements and construction safety constraints. Consequently, conventional automated assembly solutions designed mainly for regular components or single-specification workflows are difficult to reuse reliably under diverse tasks and dynamic working conditions [3].

Meanwhile, construction components are typically designed by BIM/CAD prior to on-site execution. These models include the information on geometric parameters, semantic annotations, and connectivity relationships. Transforming the model data into computationally accessible assembly constraints for robotic systems could reduce reliance on manual intervention at robot control, thereby diminishing the complexity of perception and planning while enhancing system adaptability to diverse tasks [4].

However, existing assembly systems lack mechanisms to extract connection features, mating relationships, and assembly hierarchies from BIM/CAD models. As a consequence, model information is difficult to convert into executable assembly constraints, and key assembly parameters still depend on manual specification. Moreover, even when a reasonable assembly sequence and target poses are available, the execution stage may require rewriting robotic arm control programs for different tasks, thereby limiting the efficiency and generalizability of assembly automation [5].

Recently, the capabilities of multimodal large language models (MLLMs) have provided a new solution to address the abovementioned challenges. MLLMs support multi-source data integration and semantic reasoning, enabling BIM/CAD information, on-site perception data, and task objectives to be jointly incorporated into the decision-making process, thereby producing assembly sequences and execution strategies that are aligned with task specifications [6]. Moreover, given the dispersed knowledge and highly variable task compositions in the assembly of customized components, MLLMs can generate robot control commands and reduce reliance on hard-coded workflows, thereby improving system generalizability and scalability [7].

However, directly applying MLLMs to assembly decision-making presents three challenges. Fundamentally, the core research question is how to transform scattered BIM/CAD spatial data into an executable, feedback-driven assembly process via MLLMs without compromising execution safety. First, BIM/CAD assembly constraints, including mating relations and insertion directions, are scattered across model entities rather than provided as explicit constraints. Therefore, the MLLM needs the capability to parse these structured constraints and generate reasonable assembly sequences accordingly. Secondly, assembly operations are sensitive to pose alignment and contact behaviours. Robot control commands depend on real-time feedback and fine-grained motion control. Directly generating low-level numerical control commands via MLLMs introduces severe hallucination-induced safety risks. Thirdly, assembly is a long-horizon process with strong requirements for traceability of construction progress. MLLMs need to continuously perceive the current execution state and plan subsequent actions based on it.

To address these challenges, this paper proposes a closed-loop agent framework to assembly customized building components. Firstly, the framework enforces coordinate consistency between the Rhino space and the robot space, and leveraging MLLMs understanding of the model structure, transforms BIM/CAD model data into assembly constraints for robotic control. Then, to prevent unconstrained code generation, it encapsulates Rhino functionalities and robotic capabilities as strongly typed APIs, constraining agent decisions to an executable action space to reduce hallucinated calls. Finally, the framework adopts a component-color-annotation mechanism to continuously reflect assembly progress in the visual input, and together with execution logs, form a traceable chain of evidence. Through this closed-loop mechanism, the system can achieve stable grasping, placement, and assembly execution under on-site scenarios, thereby improving the adaptability and engineering deployability of customized component assembly systems.

2 Related Works

2.1 Construction Assembly Automation

The construction assembly automation process consists of task planning, component recognition and localization, and robotic execution of assembly operations. At the task-planning level, existing studies have employed symbolic representations and assembly-constraint solving to generate assembly sequences and feasible trajectories, while incorporating reachability analysis and collision checking to ensure geometric feasibility [8]. Such approaches rely on an explicit set of computable assembly constraints. However, in construction scenarios, connection features, mating relations, and assembly hierarchies are often distributed across model entities and parameters. The lack of a unified mechanism for extraction and formal representation indicates that key assembly parameters still require manual completion.

In terms of recognition and pose estimation, existing methods mainly rely on RGB or RGB-D vision, point-cloud registration, and marker-based localization, to provide 6D pose data for grasping and placement [9]. These approaches are generally stable for regular components or in controlled and structured environments. However, they are susceptible to occlusion, illumination variation, and dust interference on dynamic, complex, and unstructured construction sites. With respect to customized components, the perception module often requires additional adaptation or recalibration, limiting cross-project reusability.

At the execution stage, assembly involves contact operations such as alignment, insertion, and surface mating. Existing studies have improved error tolerance and docking stability by leveraging force sensing, compliant control, impedance control, and contact detection, thereby improving assembly success rates [10]. However, such strategies typically require explicit specifications of connection geometry, insertion directions, and process parameters, and they often demand connection-specific control design and extensive parameter tuning. With changes to component specifications or connection relations, the cost of transferring these strategies increases substantially, and therefore, practical implementations remain highly task-specific.

2.2 MLLM-driven Assembly Automation

The inherent strengths of MLLMs provide new methodological references for assembly automation. Existing studies have adopted MLLMs to task understanding and high-level planning, generating assembly steps, action sequences, and tool-invocation plans by integrating natural-language instructions, environment states, and prior knowledge [11]. In addition, compared to rule-driven workflow orchestration, MLLMs offer greater flexibility in handling changes to task specifications and in handling dispersed knowledge. MLLMs have therefore been explored as an approach to reducing reliance on hard-coded logic and to enhancing cross-task strategy composition.

However, challenges remain when adopting MLLMs to construction assembly automation. Firstly, systematic mechanisms for transforming BIM/CAD information into assembly constraints are still lacking. Assembly-relevant information is distributed across geometry, attributes, and topological relations, such as sets of

mating surfaces, hole-shaft coaxial relations, insertion directions, and assembly-level dependencies [12]. Existing MLLM-based approaches provide priors in textual form or through limited structured inputs , relying on general scene observations rather than precise geometric data. They lack a unified pipeline for constraint extraction, formal representation, and consistency checking. Consequently, generated sequences often violate interference or dependency constraints, necessitating manual correction. This issue becomes more pronounced for complex connections or customized components.

Secondly, the reliance of contact assembly on real-time closed-loop control makes it difficult for MLLMs to directly produce executable low-level strategies. Operations such as alignment, insertion, and mating surface require compliant control, impedance parameter tuning, contact-event detection, and failure recovery [13]. They also depend on process constraints, including specified assembly tolerances, allowable pose deviations, and trigger thresholds. Existing studies typically confine MLLMs to high-level planning while delegating low-level execution to conventional controllers. However, the interface between these layers lacks clearly parameterized action primitives and pre-execution feasibility checks. Furthermore, allowing MLLMs to directly generate numerical control commands introduces significant hallucination-induced safety risks. Consequently, strategy transfer relies heavily on task-specific manual tuning.

Thirdly, existing MLLM frameworks provide insufficient support for traceable management of long-horizon assembly processes. Multi-step assembly requires continuous identification of the current status and systematic recording of decision-relevant evidence, such as constraint provenance, target poses, execution parameters, perception results, and contact feedback, so that rollback, replanning, and continued execution can be performed when necessary [14]. Current approaches focus on single-turn or single-action planning [15]. Their process-state and memory representations are fragmented and non-unified, which makes it difficult to support cross-step consistency checking and auditable traceability. Consequently, these systems are more prone to stage drift and unstable recovery under dynamic conditions, task changes, and multi-device collaboration.

3 Methodology

This paper proposes an agent-based automation framework for construction assembly operations, which integrates the model-space information from Rhino with the reasoning and command-generation capabilities of an MLLM to drive robots in executing multi-step assembly tasks. Figure 1 illustrates the framework, which comprises two modules. The first is a consistency-constrained spatial module, which establishes a spatial correspondence between Rhino model geometry and the robot operational space, and constrains the system data and action boundaries via a strongly typed API to reduce risks, incurred by inconsistent data and unsafe function calls. The second is an agent-based assembly automation module, which orchestrates iterative perception and planning, invokes a function library to perform assembly operations, and records execution evidence to support progress tracking and state updates.

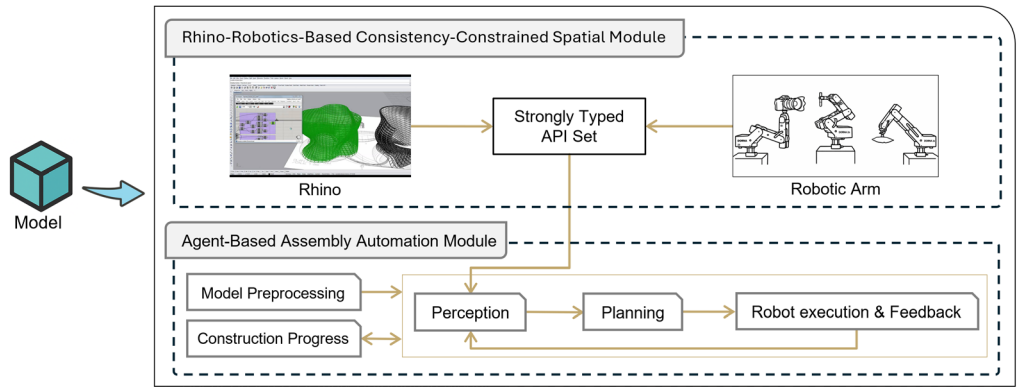

Figure 1. Framework of assembly automation in construction.

3.1 Consistency-Constrained Space

The consistency-constrained space consists of two components, namely geometric-space consistency and system-data consistency. Geometric-space consistency aligns the Rhino model geometry with the robot operational space, ensuring that assembly decision-making and motion execution are conducted within a unified coordinate frame. System-data consistency constrains the system data and action boundaries through a strongly typed API, restricting MLLM decisions to an executable and auditable scope, thereby reducing the risks associated with data inconsistency and unsafe calls.

With respect to geometric-space consistency, Figure

2 illustrates the coordinate correspondence between the Rhino space and the robot operational space. In the Rhino environment, a reference coordinate frame $Oxyz$ is defined, and a coordinate frame $O'x'y'z'$ is defined in the robot operational space. By establishing this correspondence, the coordinates of any point expressed in $Oxyz$ are kept identical to those expressed in $O'x'y'z'$, thereby avoiding additional coordinate-transformation errors between decision-making and execution.

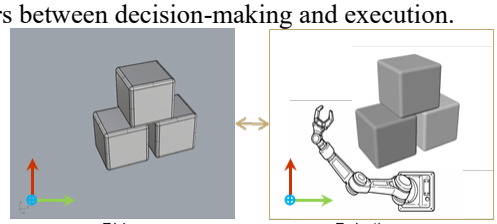

Figure 2. Coordinate consistency between Rhino and robot space.

In terms of system-data consistency, the capabilities of the Rhino and the robot are uniformly encapsulated as a strongly typed API to form an invocable function library. This library includes communication between Rhino and the robot as well as coordinate-related operations. It also includes Rhino capabilities such as model queries, extraction of geometry and parameters, view manipulation and screenshot acquisition, and component color-annotation, together with robot capabilities such as target recognition and grasping, pose control, and motion execution. Each API entry provides a clear functional description and typed input and output specifications, and defines unit conventions, valid ranges and default values, coordinate-frame and pose-representation rules, preconditions, failure return formats, and logging requirements. The agent is restricted to selecting and invoking actions within this predefined set. The API input and output domains are pre-validated and explicitly constrained, and consequently, the system can confine the decision-making of the generative model to an executable data space, thereby reducing the risk of hazardous calls induced by model hallucinations while improving data consistency and debugging traceability.

3.2 Agent-Based Assembly Automation Workflow

Based on the consistency-constrained space, the agent-based assembly automation module drives assembly tasks through a closed loop of iterative perception, planning, and execution. The core objective is to continuously obtain sufficient geometric and state information from Rhino under customized component variations and on-site uncertainties, and to generate an executable sequence of robot commands accordingly. Figure 3 shows that the workflow comprises model preprocessing, the agent closed loop, and a construction progress sub-process.

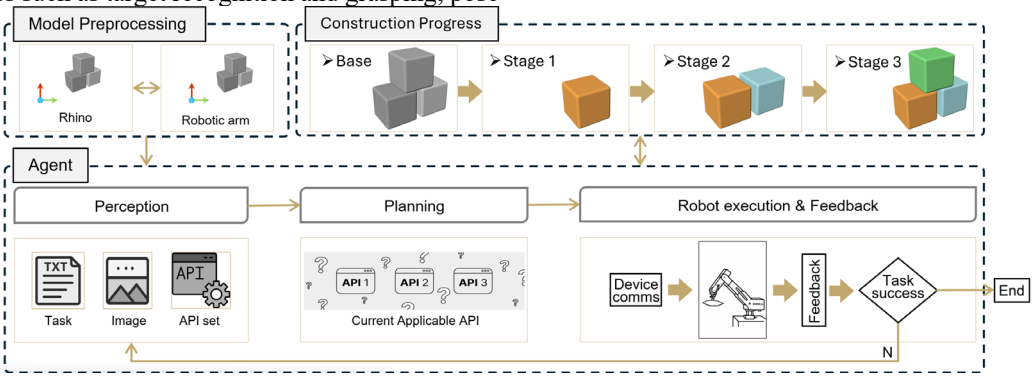

Figure 3. Workflow of the agent-based assembly automation module.

Model preprocessing is performed before entering the agent loop to establish a stable relationship among components, coordinate data, and delimit the constructible region. When only coordinate information is available, the agent is unable to accurately infer component categories and assembly relations, and therefore cannot generate a reliable assembly sequence. In contrast, when model images and the relative spatial relations among components in the scene are provided, the agent can more readily perform semantic understanding and sequencing inference. Accordingly, during preprocessing, workers specify the construction region of the model within the robot operational space and achieve global registration by adjusting the model placement under $Oxyz$. $Oxyz$ corresponds to $O'x'y'z'$, and therefore, the spatial positions and keypoint coordinates of all components can be determined with a fixed model placement. After preprocessing, the system enters the agent closed loop, enabling subsequent geometric extraction, grasp reasoning, and pose generation to be grounded on a stable spatial reference.

Within each closed-loop iteration, the agent initiates the perception stage. The agent parses the currently available APIs and the system state, and selectively invokes APIs to obtain Rhino view images, the required structured parameters, and rotated views, thereby

analyzing the task based on multimodal outputs. The process then proceeds to the planning stage. Guided by the task objective and model constraints, the agent selects the next API sequence to be invoked and generates the corresponding parameters, including the target component identifier, grasp candidates, target poses, and execution strategies. Finally, the system enters the execution stage. A communication channel between Rhino and the robot is established, and the APIs are invoked sequentially to complete the specified operations. With respect to each API call, the inputs, outputs, and runtime status are stored in logs and used as reference for the subsequent perception stage. This design enables the agent to update its decisions according to execution outcomes and, when necessary, trigger continued execution, rollback, and replanning.

The construction progress sub-process provides the agent with an explicit representation of progress, thereby supporting stage recognition and state updates. This paper adopts a visualization-based annotation approach using component coloring to distinguish assembled components from unassembled ones. After each component assembly action is completed, the system applies a color mark to the corresponding component via the "Colour" and "Preview" nodes in Grasshopper within Rhino 8. Subsequently, in the Rhino screenshots captured during the next perception stage, color-marked components indicate completed assembly, whereas uncolored components indicate pending assembly. This mechanism enables assembly progress to be continuously reflected in the visual inputs. Together with the API logs, it forms traceable evidence that facilitates stage assessment, failure recovery, and subsequent action planning.

4 Implementation and Results

Figure 4(a) shows an assembly validation scenario, established in a real physical environment. Due to limitations in the manipulator workspace and the available site scale, two scaled-down yet representative tasks, namely table assembly and façade structure assembly, were adopted as proof-of-concept cases to validate the proposed assembly agent framework. The two cases respectively represent an irregular short-horizon task and a regular long-horizon task, and are used to evaluate closed-loop executability, sequence consistency, and safety performance under different assembly characteristics. In addition, the system utility and effectiveness are evaluated from the perspectives of planning-and-execution capability and execution safety.

The system was configured with a Windows 11 workstation (NVIDIA RTX 4090) to run agent inference and decision-making, provide Rhino API, and host a unified logging module. The modeling platform employed Rhino 8 with Grasshopper for parametric model management and visual annotation. On the execution, a HansRobot E10 collaborative robotic arm was controlled by an Ubuntu 20 host. A DH-ROBOTICS AG-160-95 gripper was mounted at the end effector to support grasping and handling. The manipulator was equipped with a RealSense D455 camera to acquire RGB-D data. A target recognition and grasp candidate generation module can output object categories and candidate grasp parameters as callable functions for the agent. To ensure spatial consistency across the system, camera-to-robot frame calibration was completed before the experiments.

4.1 Case Study

Firstly, the proposed system was deployed to complete the assembly of the table shown in Figure 4(b), with the planning and execution workflow illustrated in Figure 5. The assembly started with a operator moving the model to an appropriate assembly area in the Rhino interface, after which Rhino Grasshopper extracted and stored the component poses and associated textual data. The agent then invoked the Determine the grasping target function from the capability library to identify the component to be assembled at the current step. The inputs to this function included a Rhino screenshot together with the corresponding pose and textual data. Based on a joint analysis of the image and text, the MLLM inferred that the first assembly object was a table leg. Because the prompt for this function prioritized targets that were farther from the manipulator, the output specified the table leg with ID leg2 and its corresponding pose pose2.

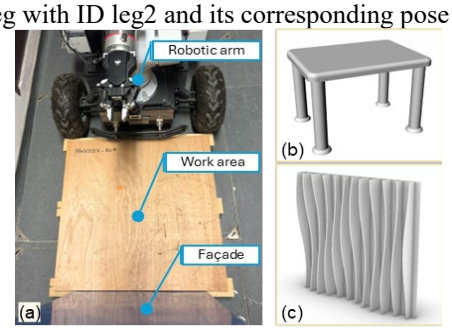

Figure 4. Experimental setup and assembly task, (a) table, (b) façade structure.

Subsequently, the agent invoked the Target recognition function to detect the table leg "leg2" in the scene. This function is a pre-validated module that fuses image-based recognition with depth camera measurements to estimate the grasp pose, denoted as grasp_pose. Building on this result, the agent further called the manipulator pick-and-place function, using the saved grasp_pose and the target component pose pose2 stored in the system memory to drive the arm to perform grasping and placement. With these steps, the first-stage

component assembly was completed. The system then issued a Rhino command to render and color the assembled component based on its ID. In the next assembly iteration, the Rhino screenshot with color annotations was provided as visual input for the MLLM as a reference to explicitly indicate the current assembly state. The corresponding pose was retrieved by querying the component ID, enabling the workflow to progress in a closed-loop manner. The planning and execution processes for the remaining four stages are shown in Figure 5, culminating in the complete assembly of the table. This case study verifies the system capability in supporting assembly tasks involving irregular structures.

Secondly, the system was applied to assemble the building façade shown in Figure 4(c), with the corresponding planning and execution workflow illustrated in Figure 6. After the worker specified the assembly region, Rhino exported and stored the component poses and associated textual data. The agent then coordinated Rhino and the robot via function calls to complete the façade assembly, with grasp pose estimation for the target components achieved by fusing fiducial marker observations with depth information. This case study validates the system capability in supporting long-horizon assembly of regular structures.

4.2 Performance evaluation and analysis

This paper evaluates the system from two dimensions, namely planning-and-execution capability and execution safety, to validate the practicality and effectiveness of the proposed framework in construction assembly tasks.

4.2.1 Planning-and-execution capability

The planning-and-execution capability evaluation examines whether the system can generate a correct assembly sequence and perform reliable grasp-and-place operations in a physical environment. Three metrics are adopted. Specifically, sequence accuracy (SA) denotes the consistency between the executed assembly order and the geometric constraints of the model, and is used to quantify the sequence planning consistency of the agent under multi-round decision-making. The execution success rate (ESR) is defined as the proportion of trials in which the system completes the end-to-end assembly workflow within a given number of tests. The placement error (PE) measures the pose deviation between the final placed pose and the target pose in successful trials, and its mean value is reported in this paper.

The results are summarized in Table 1. The system exhibited stable performance in sequence planning and achieved a fully correct sequence in the façade task. These results indicate that the progress representation mechanism based on Rhino screenshots and component color-annotations can provide sufficient semantic information for the MLLM, thereby enabling effective inference of assembly dependencies among components. In the table task, one sequence deviation was observed across 20 trials [16]. This deviation is primarily attributable to the more complex local irregular structure and tighter spatial coupling in this task, where component reachability and grasp feasibility are more sensitive to the current scene state.

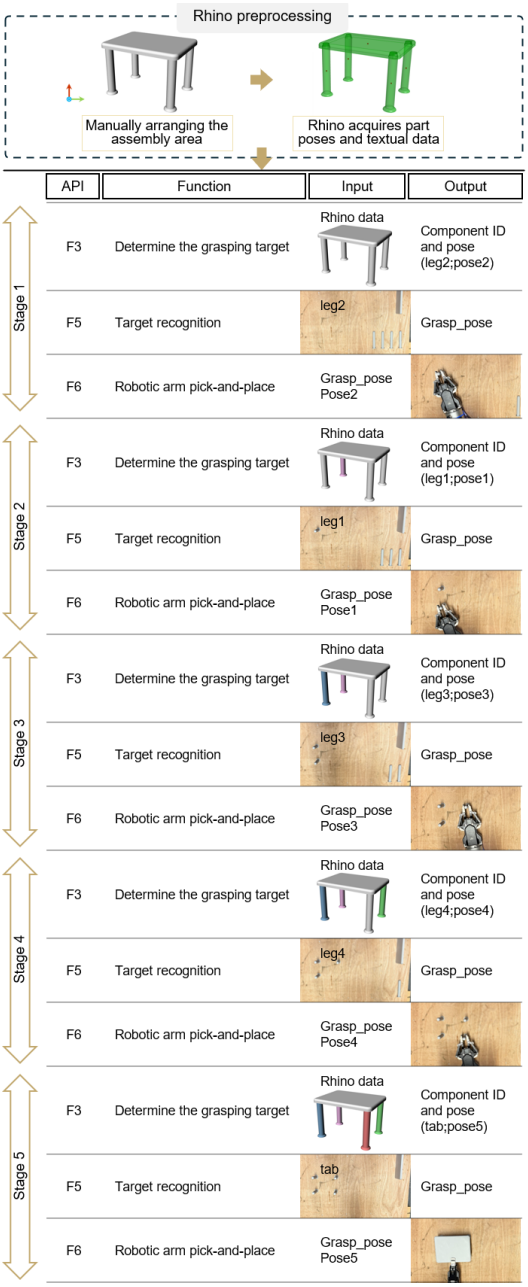

Figure 5. Planning-and-execution workflow for table assembly.

By contrast, ESR was mainly affected by execution-level factors rather than sequence planning errors. In terms of the execution success rate, failures in the table task primarily arose from insufficient grasp stability

caused by the slender geometry of the table legs, as well as collision protection being triggered during insertion due to accumulated pose errors. Failures in the façade task were mainly attributed to recognition deviations induced by fiducial marker occlusion or illumination variations. Regarding placement error, the façade task achieved higher accuracy than the table task. The higher accuracy in the façade task is attributable to the use of marker-based pose estimation for façade components, which provides higher localization accuracy than the table-leg recognition scheme based on depth-camera point-cloud registration.

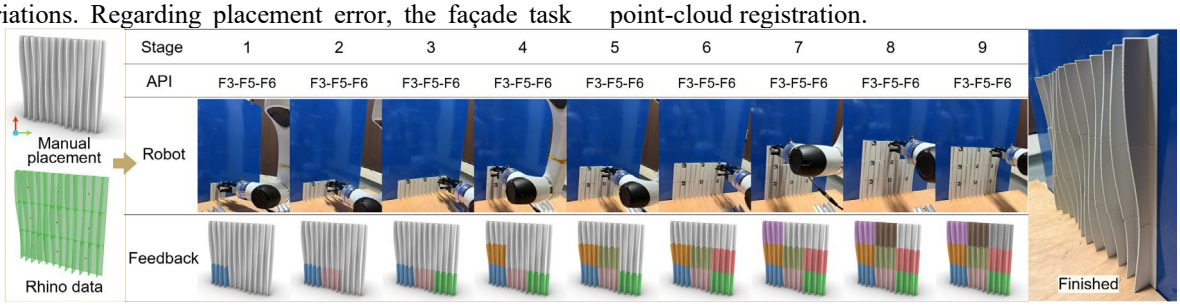

Figure 6. Planning-and-execution workflow for building façade assembly.

Table 1. Planning-and-execution performance in two assembly cases

Case	No. of Tests	SA	ESR	PE(mm)
Table	20	0.95	0.75	1.70
Façade	20	1.00	0.85	0.60

4.2.2 Execution safety

The safety evaluation aims to examine the ability of the strongly typed API constraints and feedback mechanism to reduce the risk of unsafe calls. The baseline provides only function names and brief descriptions, enabling the model to autonomously plan and generate parameter values. Configuration B introduces a strongly typed API by providing a complete specification, including input and output types and units, value ranges, coordinate-frame conventions, preconditions, and logging requirements, thereby bounding the output space of the generated plans. Configuration C corresponds to the proposed method. Building on Configuration B, it further incorporates Rhino color-based feedback, treating assembly progress as visual evidence in subsequent decision-making, which reduces phase misclassification and abnormal calls.

The results are reported in Table 2. The baseline produced a large number of calls that were non-executable or violated preconditions in both tasks, primarily due to deviations in parameter ranges, units, and coordinate-frame conventions. Configuration B increased the number of valid calls, indicating that a strongly typed API can effectively reduce the risks associated with inconsistent data and unsafe invocations. Configuration C further demonstrates that progress feedback provides stable evidence for multi-round decision-making, thereby helping maintain phase consistency and reducing erroneous API calls. One failure was still observed in the table task, which is consistent with the higher sensitivity of irregular-structure assembly to perception and contact execution. Nevertheless, overall safety was improved.

Table 2. Safety performance under different constraint configurations

Case	No. of Tests	Baseline	Strongly typed API	Proposed Method
Table	20	0.15	0.70	0.95
Façade	20	0.25	0.75	1.00

5 Discussion

The agent-based construction assembly automation framework proposed in this paper achieves a closed-loop control pipeline in a real-world physical environment, spanning from Rhino models to robotic assembly execution. Its feasibility and effectiveness are validated in twofold, namely planning-execution capability and safety performance. However, the framework still exhibits several limitations in practical applications.

In terms of assembly sequence planning, the MLLM infers assembly dependencies mainly from Rhino screenshots and component color-annotations. However, the current framework lacks explicit representations of key assembly constraints, which may limit robustness for complex connections. Future work could introduce graph-structured constraints to support more precise reasoning. In execution, contact operations in the current tasks reveal issues such as unstable grasping and insertion pose errors. Future work could encode process constraints into parameterizable action interfaces to enable the agent to adaptively plan parameters. In terms of safety, strongly typed APIs can intercept statically non-compliant calls, but they provide limited support for responding to dynamic anomalies. Future work could integrate force or contact feedback to enable online detection and recovery.

6 Conclusion

This paper proposes a closed-loop agent framework to assemble customized building components. It encapsulates Rhino and robotic capabilities into a strongly typed API library, enabling the agent to generate and execute multi-step assembly procedures within verifiable action boundaries. Real-world physical experiments were conducted using two scenarios. Results show that an SA value above 0.95 validates the system's ability to maintain consistent assembly sequence planning under closed-loop, multi-round decision-making. A safety planning rate above 0.95 confirms that, supported by strong typing constraints and progress evidence, the system can suppress non-executable and abnormal calls, providing execution safety assurance.

Future work could focus on automatically extracting assembly constraints, such as mating surfaces and hole-axis relationships, from the model and representing them in a structured form. This will reduce reliance on implicit reasoning from Rhino screenshots while preserving sequence inference capability. In addition, contact-rich operations such as alignment and insertion will be encapsulated as parameterizable actions, and force or contact feedback will be integrated to enable online detection and recovery from anomalies, including slippage, grasp failure, and insertion obstruction, thereby improving execution robustness and safety responsiveness under complex operating conditions.

Acknowledgement

The author would like to acknowledge the funding support from the Hong Kong Polytechnic University (P0051072).

References

[1] Y. Cohen, H. Naseraldin, A. Chaudhuri, and F. Pilati, "Assembly systems in Industry 4.0 era: a road map to understand Assembly 4.0," *Int. J. Adv. Manuf. Technol.*, vol. 105, no. 9, pp. 4037–4054, 2019.

[2] E. Gasparri and M. Aitchison, "Unitised timber envelopes. A novel approach to the design of prefabricated mass timber envelopes for multi-storey buildings," *J. Build. Eng.*, vol. 26, p. 100898, 2019.

[3] B. Chu, K. Jung, M. T. Lim, and D. Hong, "Robot-based construction automation: an application to steel beam assembly (Part I)," *Autom. Constr.*, vol. 32, pp. 46–61, 2013.

[4] L. Ding, W. Jiang, Y. Zhou, C. Zhou, and S. Liu, "BIM-based task-level planning for robotic brick assembly through image-based 3D modeling," *Adv. Eng. Informatics*, vol. 43, no. April 2019, p. 100993, 2020.

[5] M. Noghabaei, Y. Liu, and K. Han, "Automated compatibility checking of prefabricated components using 3D as-built models and BIM," *Autom. Constr.*, vol. 143, p. 104566, 2022.

[6] H. Luo, J. Wu, J. Liu, and M. F. Antwi-Afari, "Large language model-based code generation for the control of construction assembly robots: a hierarchical generation approach," *Dev. Built Environ.*, vol. 19, p. 100488, 2024.

[7] H. Pu, X. Yang, J. Li, and R. Guo, "AutoRepo: a general framework for multimodal LLM-based automated construction reporting," *Expert Syst. Appl.*, vol. 255, p. 124601, 2024.

[8] N. Chokshi, "Robot conquers one of the hardest human tasks: Assembling Ikea furniture," *New York Times*, vol. 18, 2018.

[9] J. A. Marvel, R. Bostelman, and J. Falco, "Multi-robot assembly strategies and metrics," *ACM Comput. Surv.*, vol. 51, no. 1, pp. 1–32, 2018.

[10] K. H. Petersen, N. Napp, R. Stuart-Smith, D. Rus, and M. Kovac, "A review of collective robotic construction," *Sci. Robot.*, vol. 4, no. 28, p. eaau8479, 2019.

[11] Y. Zheng *et al.*, "PlanAgent: a multi-modal large language agent for closed-loop vehicle motion planning," *arXiv Prepr. arXiv2406.01587*, 2024.

[12] G. Gorjup, G. Gao, A. Dwivedi, and M. Liarokapis, "A flexible robotic assembly system combining cad based localization, compliance control, and a multi-modal gripper," *IEEE Robot. Autom. Lett.*, vol. 6, no. 4, pp. 8639–8646, 2021.

[13] Z. Chen and A. Adel, "Advancing robotic assembly in construction: Innovations, challenges, and opportunities," *Autom. Constr.*, vol. 178, p. 106370, 2025.

[14] F. Suárez-Ruiz, X. Zhou, and Q.-C. Pham, "Can robots assemble an IKEA chair?," *Sci. Robot.*, vol. 3, no. 17, p. eaat6385, 2018.

[15] P. Cao *et al.*, "Large Language Models for Planning: A Comprehensive and Systematic Survey," *arXiv Prepr. arXiv2505.19683*, 2025.

[16] L. Faulkner, "Beyond the five-user assumption: benefits of increased sample sizes in usability testing," *Behav. Res. Methods, Instruments, Comput.*, vol. 35, no. 3, pp. 379–383, 2003.