VLM-Based Component-Level Task Planning for Adaptive Bricklaying: Integrating 3D Models and 2D Shop Drawings

Nayun Kim[1,2], Florian Noichl[1,2] and André Borrmann[1,2]

[1]Chair of Computing in Civil and Building Engineering,
Technical University of Munich, Germany
[2]TUM Georg Nemetschek Institute, Germany

nayun.kim@tum.de, florian.noichl@tum.de, andre.borrmann@tum.de

Abstract -

Construction robots require component-level task specifications that include per-unit dimensions, 6D poses, and assembly sequences respecting construction constraints. However, existing BIM-based robot task planning pipelines rely on either ad-hoc heuristic decomposition or exhaustive pre-modeling, both of which are brittle under design changes and fail to capture the assembly semantics encoded in 2D shop drawings. We propose a two-phase framework that integrates design-stage 3D models and 2D shop drawings to generate component-level task specifications for downstream robotic execution. The framework separates task formalization from task compilation, with two intermediate representations (IRs) serving as the interface between phases. In the formalization phase, a Vision-Language Model (VLM) extracts assembly logic from 2D shop drawings into an Assembly Representation (AsmRep), while geometric constraints are obtained from a design-stage IFC 3D model as a Building Element Representation (ElemRep). In the compilation phase, a deterministic IR compiler integrates both representations to generate robot-executable task specifications, including per-unit 6D poses, dimensions, and assembly sequences. Experiments on bricklaying demonstrate that foundation VLMs can reliably formalize drawing-encoded rules such as layout lines, anchor conventions, and bond patterns, and that varying only the AsmRep under identical ElemRep produces different decompositions and sequences. A preliminary simulation study further shows that the generated task specification can be consumed by a downstream robotics layer. These results support the feasibility of the proposed framework and its integration with downstream robotic execution layers.

Keywords -

Assembly Sequence Planning; Robot Task Planning; VLM; Construction Robotics; Bricklaying;

1 Introduction

Construction robots require component-level task specifications that are executable on site: the dimensions and 6D pose of each unit, and an assembly order that respects construction constraints. Unlike factory automation, construction is executed in a changing environment: dimensions drift, tolerances accumulate, and design details evolve across phases and trades [1]. As a result, an assembly specification is not just an order of operations; it encodes where work is referenced (e.g., layout lines), how boundary conditions are handled, and how residual errors are absorbed.

Building Information Modeling (BIM) is the dominant digital representation for 3D building geometry in construction. Therefore, many construction robotics studies start from BIM to derive executable task specifications, such as decomposing assemblies, assigning placements, and generating sequences, directly from 3D models. However, geometry-centric planning often shows a limitation in applying to the practice. Existing pipelines either use ad-hoc heuristic decomposition[2, 3], which is rigid and hard to generalize, or rely on exhaustive pre-modeling and manual scripting [4, 5, 6], which is labor-intensive and brittle under inevitable revisions. Crucially, both strategies face practical limitations on large-scale assemblies and dynamic sites. As the number of components increases, these approaches become increasingly difficult to maintain, and they remain brittle under design revisions and site deviations because the required assembly semantics is rarely encoded explicitly.

Construction tasks are developed progressively across project phases by multiple stakeholders. To communicate the maturity and intended use of model elements, the Level of Development (LOD) specification is widely used. LOD 300 typically provides measurable design-stage geometry (size, shape, location, and orientation), while LOD 400 represents the detailed components for fabrication and installation in 3D models [7]. However, the procedural knowledge required for robot execution, such as assembly logic, reference conventions, and sequencing, is rarely encoded explicitly in 3D BIM models. Instead, this semantic information is commonly delivered to the field through 2D shop drawings [8]. While shop drawings are a primary carrier of such semantics, they express it implicitly through

symbolic, human-centric drafting conventions rather than explicit procedural rules, making them not directly usable for generating per-unit placements and sequential robot actions.

Recent progress in vision-language models (VLMs) suggests a path to interpret such drawing content [9, 10, 11]. Yet, using VLMs for long-horizon geometric planning remains unreliable and difficult to constrain, especially when the desired output must be consistent, verifiable, and reproducible at the component level [12]. This motivates a different role for VLMs in construction robotics: not as end-to-end planners, but as formalizers that extract and structure drawing-encoded assembly semantics into a constrained intermediate representation (IR).

These observations raise three research questions: **(RQ1)** What information from design-stage 3D models and 2D shop drawings is necessary for robot assembly, and how can it be structured into intermediate representations? **(RQ2)** How can VLMs reliably extract implicit assembly semantics from 2D shop drawings? **(RQ3)** How can extracted semantics be compiled deterministically into robot-executable task specifications?

To address these questions, we present a two-stage pipeline that separates (1) **task formalization**, which extracts assembly rules from 2D shop drawings and geometric constraints from design-stage 3D models and encodes them into two intermediate representations: the Assembly Representation (AsmRep) and the Building Element Representation (ElemRep); and (2) **task compilation**, in which a deterministic compiler compiles both IRs into a robot-executable assembly task specification (per-unit 6D poses, dimensions, and assembly sequences). Experiments on bricklaying show that, under identical BIM-driven geometry, varying only assembly rules produces different decompositions and sequences. This indicates that our system explicitly captures assembly logic, enabling scalable component-level task planning and providing a foundation for rapid re-planning in a dynamic construction environment.

2 Related works

Existing BIM-based robotics pipelines largely treat task planning as geometric translation from 3D models. Early work mapped IFC to robotic description formats to reconstruct the building environment for robot operation [5]. Subsequent studies attempted to derive component-level information: Brinkhoff and Kolani [2] geometrically slices LOD 300 walls into brick via scripts, Gao et al. [3] generated assembly coordinates with heuristic spatial sorting, and Wong Chong et al. [4] extracted coordinates for wood frame assembly from BIM. To improve site adaptability, Oyediran et al. [13] integrated 4D BIM scheduling and Fingrut et al. [14] employed computer vision for adaptive bricklaying correction. More recently, Kim et al. [6] used an LLM for dialogue-based robot planning, where BIM-derived context serves as a world model.

Despite these advancements, these BIM-based planning efforts demonstrate critical limitations in scalability and robustness. Explicitly modeling every component at an LOD 400 level becomes practically impossible as the scale of construction increases due to massive computational and modeling overhead. Conversely, heuristic-driven decomposition from LOD 300 models lacks the robustness needed for complex geometric scenarios, such as intricate joints or openings. These limitations motivate extracting assembly semantics from 2D drawings and compiling it deterministically with 3D geometry, rather than treating planning as geometric translation alone.

In parallel, VLMs have been applied to interpret assembly instructions in robotics. Manual2Skill [9] extracts hierarchical assembly graphs from furniture manuals to acquire robotic skills, VLM-MSGraph [10] constructs multi-hierarchical scene graphs to guide assembly sequences, and Kyaw et al. [11] utilized VLMs for component decomposition of 3D meshes. However, these systems primarily operate on general-purpose reasoning, such as identifying that a chair requires legs, and typically rely on unique identifiers for a small set of distinct parts. Such methods do not scale to repetitive construction tasks, which involve the massive repetition of identical units where assigning unique IDs to every component is infeasible. Furthermore, interpreting construction drawings requires sophisticated spatial reasoning across multiple views (plans, sections, elevations) and conventions that exceed the general functional cues used in furniture or toy assembly. Moreover, end-to-end VLM planning remains unreliable for long-horizon reasoning; recent work suggests VLMs are more effective as formalizers that translate multimodal inputs into structured representations [12].

These observations motivate our approach: formalized assembly semantics from 2D drawings via constrained VLM extraction and compile them deterministically with 3D model geoemtry.

3 Methodology

We propose a component-level assembly sequence planning framework that synthesizes geometric contexts from 3D Models with assembly rules from 2D architectural drawings to generate robot-executable assembly information.

As illustrated in Figure 1, the framework operates in two phases: (1) **task formalization**, where different levels of construction data are parsed into two structured intermediate representation (IRs), a Building Element Representation (ElemRep) encoding spatial constraints, and

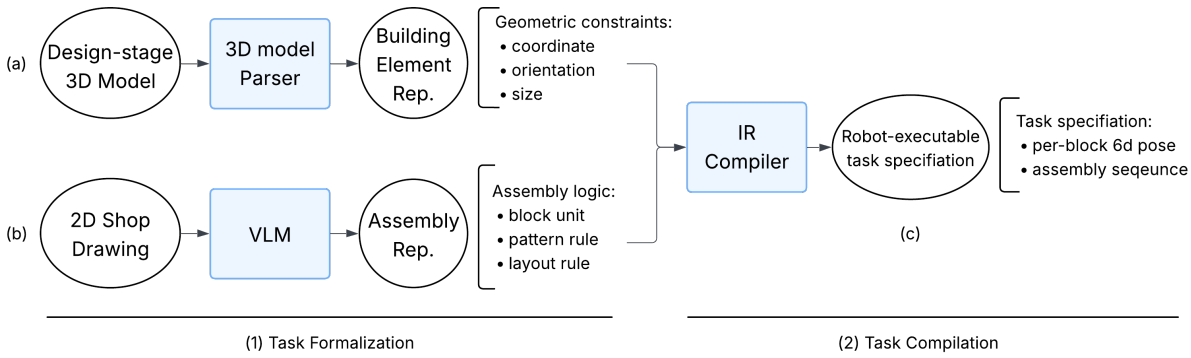


Figure 1. System overview. We separate (1) task formalization into intermediate representations (IRs): (a) Building Element Representation (ElemRep) from design-stage 3D models and (b) Assembly Representation (AsmRep) from 2D shop drawings, and (2) task compilation into (c) robot-executable assembly information via an IR compiler.

an Assembly Representation (AsmRep) encoding assembly rules; and (2) **task compilation**, where a deterministic compiler translates these IRs into a final component assembly sequence, including per-unit 6D poses, dimensions, and assembly sequence. This separation leverages the VLM's strength in semantic formalization while avoiding its unreliability in long-horizon geometric planning.

3.1 Task Formalization from 3D Models

Derived from a design-stage 3D model, the ElemRep encapsulates the geometric location and boundary of the element. We formally define the ElemRep G as a tuple comprising three primary components:

$$\mathcal{G} = (E, O, R) \tag{1}$$

- **Target Element** (E): This component defines the volumetric element to be constructed (e.g., the wall). It is represented as $E = (\mathbf{p}_e, \mathbf{s}_e)$, where $\mathbf{p}_e \in SE(3)$ denotes the global 6D pose of the element's origin, and $\mathbf{s}_e \in \mathbb{R}^3$ represents its volumetric dimensions (length, height, thickness).
- **Opening Entities** (O): This set captures geometric definitions for voids within the target element, defined as $O = \{o_1, \ldots, o_n\}$. Each opening o_k is a tuple $(\mathbf{p}_{o,k}, \mathbf{s}_{o,k})$, specifying the global pose and dimensions required for boolean subtraction.
- **Reference Alignment** (R): When the target element is constrained by an adjacent structure, this component provides the spatial referencing data. It is defined as $R = (\mathbf{c}_{ref}, \mathbf{v}_{ref}, d_{offset})$, containing the reference centroid $\mathbf{c}_{ref} \in \mathbb{R}^3$, the normalized axis direction $\mathbf{v}_{ref}$, and the scalar offset distance d_{offset} relative to the target element.

3.2 Task Formalization from 2D Drawings

Derived from a 2D shop drawing via VLM parsing, the AsmRep S captures the explicit assembly rules for interior masonry. We define $\mathcal{S}$ as a tuple comprising:

$$\mathcal{S} = (B, P, L) \tag{2}$$

- **Block** (B): Identifies the physical attributes of the finishing unit. It is represented as $B = (\mathbf{d}_{unit}, id)$, where $\mathbf{d}_{unit} \in \mathbb{R}^3$ denotes the nominal dimensions (l_u, w_u, h_u) and id specifies the material type.
- **Pattern** (P): Specifies the topological arrangement of units. It is represented as $P = (\tau, \alpha, \mathbf{d}_{joint})$, where τ denotes the bond type (e.g., running bond, stack bond), $\alpha \in [0, 1)$ is the row-wise offset ratio, and $\mathbf{d}_{joint}$ defines the visual joint thickness.
- **Layout** (L): Defines the reference axis and boundary conditions for grid distribution. It is represented as $L = (\lambda_{align}, \lambda_{anchor}, \delta, C_{edge})$. The first three parameters deterministically resolve the pattern origin: λ_{align} selects identifies the semantic reference layout line (e.g., wall center, opening center, structure finish face), δ applies a scalar offset distance from this layout line to account for finish thickness or design shifts, and λ_{anchor} determines which specific feature of the unit (e.g., brick center or edge, joint center) locks onto the layout line. Finally, to manage dimensional remainder at boundaries, C_{edge} defines the edge treatment policy $(\mathbf{s}_{min}, J_{range}, \epsilon_{adj})$, specifying the minimum allowable cut size $\mathbf{s}_{min}$ and the permissible joint adjustment range J_{range} for absorbing cumulative residuals.

3.3 Task Compilation via IR Compiler

Given ElemRep $\mathcal{G}$ and AsmRep $\mathcal{S}$, the compiler deterministically computes a robot-executable **task specification** $\mathcal{T}$:

$$\mathcal{T} = (\mathcal{U}, \pi) \tag{3}$$

where $\mathcal{U} = \{u_i\}_{i=1}^{n}$ is **the set of component units** and $\pi = (i_1, \ldots, i_n)$ is a permutation of $\{1, \ldots, n\}$ that defines **the assembly sequence**. Each unit is defined as:

$$u_i = (\mathbf{p}_i \in SE(3),\ \mathbf{s}_i \in \mathbb{R}^3) \tag{4}$$

with 6D pose $\mathbf{p}_i$ and dimensions $\mathbf{s}_i$. All decisions follow fixed rules, so identical ElemRep and AsmRep $(\mathcal{G}, \mathcal{S})$ yield an identical task specification $\mathcal{T}$.

The IR compiler first maps the target element E and openings O from ElemRep $\mathcal{G}$ into a wall-local coordinate frame defined by the wall pose in E. All layout decisions are then made in the wall-local (x, z) plane, where x follows the wall direction and z corresponds to course height. Finally, each unit extent is lifted back to a global 6D pose in $SE(3)$ using the inverse wall transform, and an assembly sequence π is assigned using a deterministic ordering rule.

Step 1: Coordinate framing From the ElemRep $\mathcal{G}$, the compiler reads the global 3D pose and dimensions of the target element E and each opening $o \in O$, and transforms them into a common wall-local coordinate frame defined by the wall pose in E. In this wall-local frame, the compiler derives the wall bounds in the (x, z) domain, where x follows the wall direction and z corresponds to course height, and maps openings into the same domain.

Step 2: Layout line and course grid setup Using the layout term $L = (\lambda_{\text{align}}, \lambda_{\text{anchor}}, \delta, C_{\text{edge}})$ from the AsmRep $\mathcal{S}$, the compiler deterministically resolves a horizontal reference layout line x_{ref}. If $\lambda_{\text{align}} =$ *WALL_CENTER*, x_{ref} is obtained from the wall extents in E; if $\lambda_{\text{align}} =$ *OPENING_CENTER*, x_{ref} is obtained from the designated opening geometry in O; and if $\lambda_{\text{align}} =$ *STRUCTURE_FINISH_FACE*, x_{ref} is obtained from the referenced structure face and offset in R. Finally, λ_{anchor} selects the anchored unit feature (e.g., unit center or edge), which determines a unique grid origin x_0. The scalar offset δ then shifts x_{ref} to account for finish thickness or design intent.

The compiler then defines the horizontal module $m_x = l_u + j_h$ from the unit length in B and head-joint thickness in P, and the vertical module $m_z = h_u + j_b$ from the unit height in B and bed-joint thickness in P. Vertically, courses are generated from the wall base upward at increments of m_z without vertical cutting.

Step 3: Pattern instantiation For each course, the bonding pattern $P = (\tau, \alpha, \mathbf{d}_{\text{joint}})$ deterministically defines a row-wise horizontal offset (e.g., alternating offsets for running bond). The compiler then instantiates nominal 1D unit spans along x, anchored at the grid origin x_0 with the course-dependent offset. Depending on λ_{align}, the compiler uses either *grid mode* for center-aligned reference lines (wall/opening center), where tiling is symmetric about x_{ref} and clipped to $[x_{\min}, x_{\max}]$, or *linear mode* for face-aligned reference lines (structure finish face), where tiling proceeds in a fixed direction from x_{ref} and is clipped to the same bounds. This step yields per-course nominal spans prior to applying the edge policy.

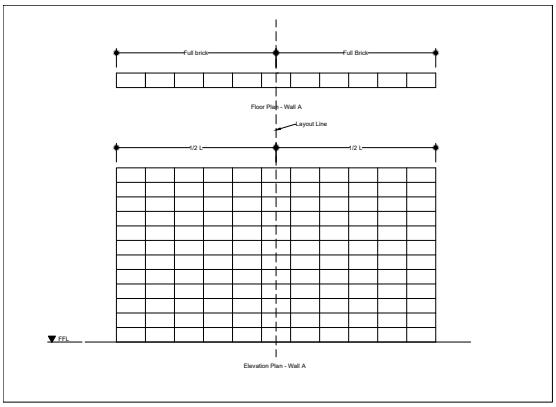
Step 4: Edge policy The compiler then applies the edge policy $C_{\text{edge}} = (\mathbf{s}_{\min}, J_{\text{range}}, \epsilon_{\text{adj}})$ to handle wall-end remainders ρ (remaining length after clipping, with head-joint thickness j_h from P): if $\rho \le \epsilon_{\text{adj}}$, the residual is absorbed by adjusting head joints within J_{range}; if $\rho \ge j_h + \mathbf{s}_{\min}$, a cut unit is appended; if neither condition holds (i.e., $\epsilon_{\text{adj}} < \rho < j_h + \mathbf{s}_{\min}$), the compiler redistributes the layout over the preceding k neighboring units to resolve the remainder. Here, k is chosen deterministically to remove ambiguity.

Step 5: Opening subtraction Openings O extracted from $\mathcal{G}$ are then subtracted from the generated spans: the compiler splits any course overlapping an opening at its boundaries and removes interior segments. Resulting fragments are checked against the minimum brick size $\mathbf{s}_{\min}$. Segments meeting this constraint are retained as cut units, while shorter ones are dropped.

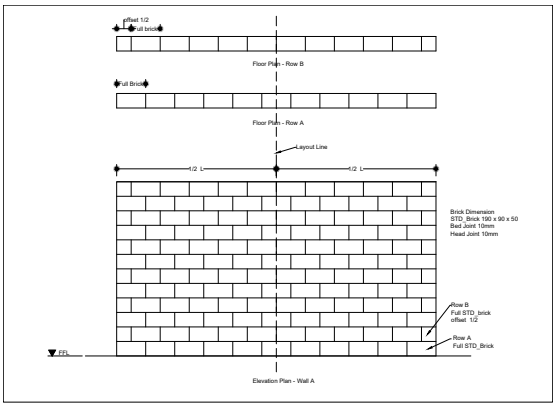
Step 6: Sequencing Finally, the compiler assigns an assembly sequence π: units are placed bottom-to-top by course index, and within each course, in increasing distance from the reference layout line. This places units near the reference line first, letting tolerances be absorbed at the boundaries instead of accumulating toward the layout line.

4 Experiments

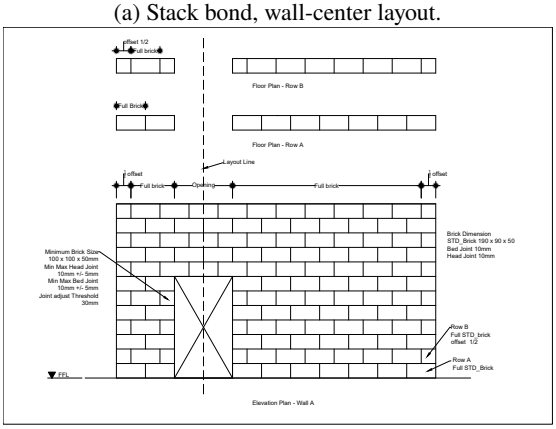
We evaluate the proposed pipeline on controlled bricklaying scenarios to test our two-phase system. First, we measure whether a VLM foundation model can formalize assembly semantics in 2D drawings into an AsmRep, with emphasis on visually implied rules (e.g., layout line, anchor, and bond type). Second, we perform a compiler sensitivity study: We fix geometry (ElemRep) and vary assembly semantics (AsmRep) to see how assembly semantics affect the decomposition and sequencing of task specification.



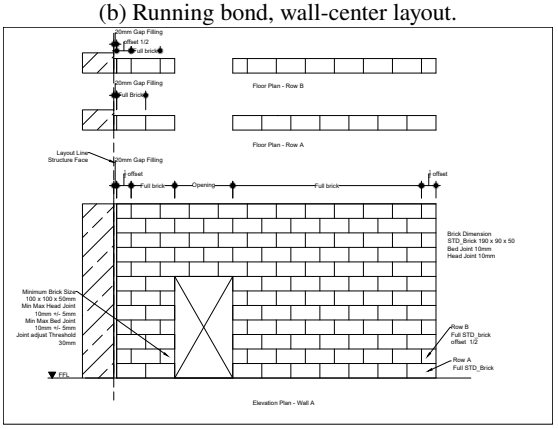
(a) Stack bond, wall-center layout.



(b) Running bond, wall-center layout.



(c) Running bond with an opening and opening-center layout.



(d) Running bond with an opening and a finish-face layout.

Figure 2. The syntactic drawings for experiment 1: Bricklaying drawing variants used for AsmRep extraction.

Table 1. Field-wise VLM extraction accuracy across drawing types and models. (k correct/n valid runs)

Field	Type	(a) stack Gemini 3	(a) stack GPT-4o	(b) running Gemini 3	(b) running GPT-4o	(c) running_opening Gemini 3	(c) running_opening GPT-4o	(d) running_opening_ref Gemini 3	(d) running_opening_ref GPT-4o
bond_type	Cat.	10/10	0/10	10/10	10/10	10/10	10/10	10/10	10/10
offset_ratio	Num.	10/10	0/10	10/10	10/10	10/10	10/10	10/10	10/10
head_joint	Num.	–	–	10/10	10/10	10/10	10/10	10/10	10/10
bed_joint	Num.	–	–	10/10	10/10	10/10	10/10	10/10	10/10
layout_line	Cat.	10/10	10/10	10/10	10/10	10/10	10/10	10/10	10/10
anchor_type	Cat.	10/10	10/10	10/10	10/10	10/10	10/10	10/10	10/10
brick_dims	Num.	–	–	10/10	10/10	10/10	10/10	10/10	10/10
opening_detected	Cat.	–	–	–	–	10/10	10/10	10/10	10/10
min_brick_size	Num.	–	–	–	–	10/10	10/10	10/10	10/10
joint_threshold	Num.	–	–	–	–	10/10	10/10	10/10	10/10
offset_distance	Num.	–	–	–	–	–	–	10/10	7/10

4.1 Experiment 1: VLM AsmRep Extraction

4.1.1 Setup

We evaluate whether foundation VLM can reliably formalize drawing-encoded assembly semantics into the schema-constrained AsmRep. We compare Gemini 3 with GPT-4o under an identical schema and prompting setup. Both models were configured with temperature set to 0 and structured JSON output mode enforced via a Pydantic Schema that defines three field groups: brick definition, pattern logic, and wall mapping rule. Each model was executed 10 times independently. We report field-level extraction accuracy as the number of runs producing the correct value for each AsmRep field, evaluated against a manually defined ground truth per drawing. The full prompt specification is available in the git repository. [1]

We created four synthetic brickwork drawings (Figure 2) with progressively increasing semantic complexity to isolate specific extraction challenges: (a) stack bond with wall-center layout, containing only bond pattern and layout line without explicit numeric parameters; (b) running bond with wall-center layout, introducing offset logic and joint dimensions; (c) running bond with an opening and opening-center layout, adding opening detection and edge-policy parameters; and (d) running bond with an opening

[1] https://github.com/narchitect/assembly-seqeunce

and structure finish-face layout, further adding a reference entity and offset distance. This progression allows us to evaluate VLM reliability as drawing complexity and the number of extractable semantic fields increase.

4.1.2 Result

Table 1 reports field-level extraction accuracy across 10 independent runs per configuration. Gemini 3 achieved perfect extraction across all applicable fields and all four drawings. GPT-4o performed comparably on drawing (c) and (d) with running bonding, but failed on implicit visual conventions in the stack bonding drawing (a). It misidentified stack bond as running bond in all 10 runs and also returned null for offset distance in 3/10 runs, confirming that VLM outputs are not fully deterministic even at temperature 0. These results suggest that the primary bottleneck in extraction is not numeric recovery but the interpretation of visually encoded drafting conventions, such as bond type and offset logic.

4.2 Experiment 2: IR Compiler Task Compilation

4.2.1 Setup

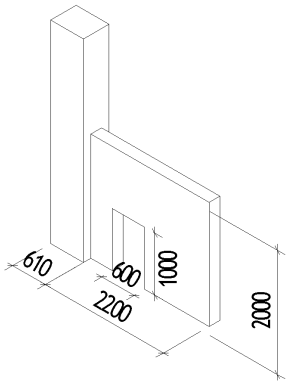


Figure 4. A sample design-stage 3D model for IR compilation experiment

For task compilation, the IR compiler takes both an ElemRep and an AsmRep as inputs. We construct the ElemRep by exporting a design-stage 3D model to IFC (Figure 4) and parsing it with IfcOpenShell to extract the wall, a rectangular opening, and an adjacent reference structure, including their global 6D poses and dimensions. This ElemRep is reused across all runs.

We compile four AsmRep configurations against this identical geometry to examine how assembly semantics affect task specification generation. Configuration (a) and (b) differ in bond pattern (stack bond vs. running bond) under the same wall-center layout condition, allowing us to observe the effect of bond type on decomposition. Configurations (b), (c), and (d) share the same running bond but differ in layout line (wall center, opening center, and structure finish face), allowing us to isolate the effect of layout semantics under a fixed bond pattern.

4.2.2 Result

Figure 3 shows that, under an identical geometry (ElemRep), changing only the assembly semantics (bonding type and layoutline) term in the AsmRep produces different brick decompositions and assembly sequences.

Comparing configurations (a) and (b), the bond pattern changes the decomposition substantially. Stack bond (Figure 3a) maintains the same grid phase across courses and therefore produces more regular opening-adjacent fragmentation. In contrast, running bond (Figure 3b) introduces alternating row offsets that interact with opening boundaries, producing more varied cut-brick sizes and void patterns around jambs and lintels.

Within the running-bond configurations, Figures 3b, 3c, and 3d show that layout line alone also affects decomposition and sequencing. In the wall-center layout (Figures 3b), the overall pattern is symmetric with respect to the wall, but small fragments can appear near the opening boundaries. In the opening-center layout (Figure 3c), the bricks are arranged more symmetrically around the opening, although this often leads to uneven cuts at the wall ends. In the structure-finish-face layout (Figure 3d), the brick pattern is aligned with the referenced structure, and the remaining adjustment is shifted to the opposite side. These differences also appear in the generated sequence, because the bricks are placed in a fixed order by course and then by distance from the reference line.

4.3 Preliminary Simulation Validation

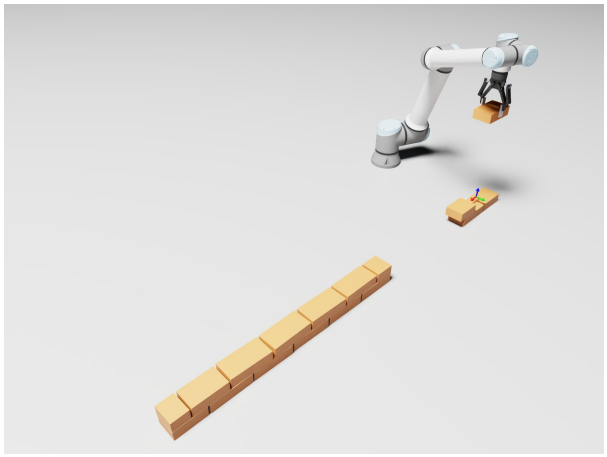
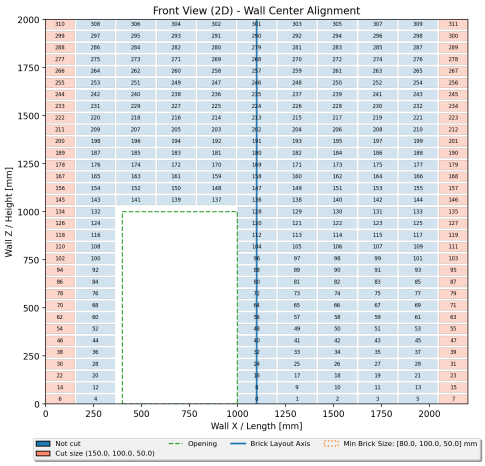
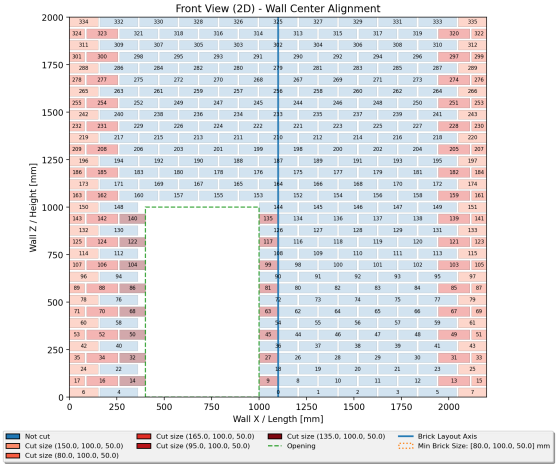


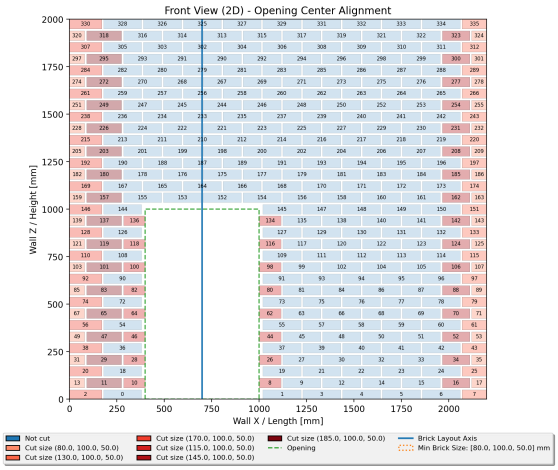
Figure 5. Compiled task specification executed in Isaac Sim: UR10e sequentially placing bricks according to the generated per-unit targets



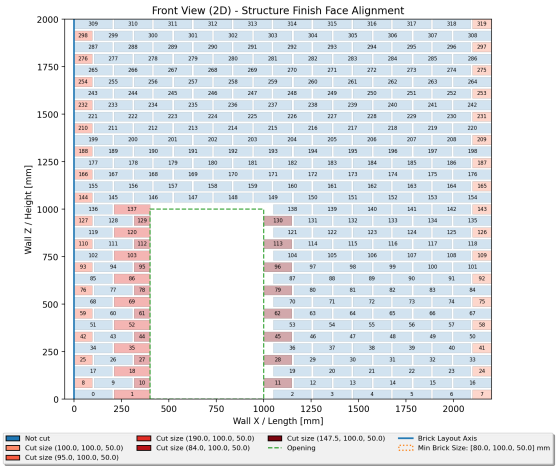
(a) Stack bond, wall-center layout.



(b) Running bond, wall-center layout.



(c) Running bond, opening-center layout.



(d) Running bond, finish-face layout.

Figure 3. The results of experiment 2: Compiled task specifications for four different AsmRep configurations under identical ElemRep. red color: cut bricks; numbers: sequence IDs

To provide an execution-level validation of the compiled task specification, we integrated the generated output into a robot simulation environment. The simulation was set up in NVIDIA Isaac Sim with a UR10e manipulator and a Robotiq 2F-140 gripper. RMPflow was used for collision-aware motion generation. A finite-state pick-and-place controller sequentially executed the per-unit targets from the compiled task specification, following an approach $\rightarrow$ grasp $\rightarrow$ lift $\rightarrow$ move $\rightarrow$ place $\rightarrow$ release $\rightarrow$ retreat cycle. As shown in Figure 5, the robot can execute sequential brick placements at the compiled target poses in simulation. Although this does not yet constitute a full validation of autonomous bricklaying, it shows that the proposed task specification can be consumed by a downstream robotics execution layer.

5 Discussion

The experiments support our central premise that assembly semantics encoded in drawings affects component-level decomposition and sequencing even under identical BIM geometry. This motivates the proposed separation between semantic formalization and deterministic compilation: VLMs translate drawing conventions into a constrained AsmRep, while the IR compiler produces reproducible, verifiable per-unit task specifications by integrating the semantic IR with the geometric IR.

RQ1 (IR specification): The results (Figure 3) indicate that the combination of an ElemRep (element bounds, openings, and optional reference structures) and an AsmRep (layout line, anchor convention, bond pattern, and edge policy) is sufficient to generate component-level brick poses, dimensions, and assembly sequences, in the evalu-

ated cases.

RQ2 (VLM Task Formalization): The results of Experiment 1 (Table 1) show that both Gemini 3 and GPT-4o perform well overall under the evaluated drawing conditions, but GPT-4o is less consistent in extracting implicit visual assembly semantics. This suggests that the main bottleneck in VLM-based semantic extraction is not the recovery of explicit numeric or textual information, but the interpretation of visual conventions such as bond pattern, layout line, and anchor type. Although the current study is limited to synthetic drawings with controlled conventions, the results indicate that foundation VLMs already have practical potential to formalize construction-relevant drawing semantics without task-specific fine-tuning in this controlled setting.

RQ3 (Deterministic compilation): Experiment 2 (Figure 3) shows that, under an identical geometry (ElemRep), varying the assembly semantics (AsmRep) changes the resulting decomposition and assembly sequence in predictable ways. The comparison between stack bond and running bond shows that bond pattern alone affects row-phase behavior and opening-adjacent fragmentation. Within the running-bond cases, changing only the layout line further changes the grid phase relative to the opening, which leads to different cut-brick sizes, void patterns, and sequences under the same minimum cut-length rule. These results show that the compiler makes the effect of assembly semantics explicit and reproducible, rather than leaving it implicit in geometric modeling alone.

A preliminary robot simulation validation further showed that the compiled task specification can be integrated with a downstream robotics layer in simulation. The present study should be read as a controlled feasibility study rather than a robustness benchmark. Within the bricklaying domain, our results show that explicit modeling of assembly semantics and compilation logic enables structured and reproducible task specification generation across controlled layout variations and edge conditions. Future work should evaluate the framework on real drawings, more complex bricklaying cases, and different assembly tasks.

6 Conclusion

This paper presented a two-phase framework for component-level task planning in adaptive bricklaying by integrating design-stage 3D models with 2D shop drawings. The main contributions are two intermediate representations, ElemRep and AsmRep, and a deterministic IR compiler that combines them to generate per-unit brick poses, dimensions, and assembly sequences in a reproducible manner. The results show that foundation VLMs can formalize key assembly semantics from drawings, and that varying assembly semantics under identical geometry leads to different decompositions and sequences. A preliminary robot simulation further showed that the generated task specification can be integrated with a downstream robotics layer.

7 Acknowledgments

We gratefully acknowledge the support of the German Research Foundation (DFG) for funding the project under grant FOR 5672, subproject A "Generic framework for simulating and executing robotized construction", grant number 517965147, as well as the support of the TUM Innovation Network "CoConstruct".

References

[1] J. Buchli, M. Giftthaler, N. Kumar, M. Lussi, T. Sandy, K. Dörfler, and N. Hack. Digital in situ fabrication - challenges and opportunities for robotic in situ fabrication in architecture, construction, and beyond. *Cement and Concrete Research*, 112:66–75, 2018. doi:10.1016/j.cemconres.2018.05.013.

[2] M. J. S. Brinkhoff and M. R. Kolani. Deriving fabrication information from BIM for automated robotic task planning. In *Proceedings of Forum Bauinformatik*. Technische Universität Hamburg, 2024. doi:10.15480/882.13543.

[3] Y. Gao, J. Meng, J. Shu, and Y. Liu. BIM-based task and motion planning prototype for robotic assembly of COVID-19 hospitalisation light weight structures. *Automation in Construction*, 140: 104370, 2022. doi:10.1016/j.autcon.2022.104370.

[4] O. Wong Chong, J. Zhang, R. M. Voyles, and B.-C. Min. BIM-based simulation of construction robotics in the assembly process of wood frames. *Automation in Construction*, 137:104194, 2022. doi:10.1016/j.autcon.2022.104194.

[5] K. Kim and M. Peavy. BIM-based semantic building world modeling for robot task planning and execution in built environments. *Automation in Construction*, 138:104247, 2022. doi:10.1016/j.autcon.2022.104247.

[6] K. Kim, P. Ghimire, and P.-C. Huang. Framework for LLM-Enabled Construction Robot Task Planning: Knowledge Base Preparation and Robot–LLM Dialogue for Interior Wall Painting. *Robotics*, 14(9):117, 2025. doi:10.3390/robotics14090117.

[7] BIM Forum. LOD Specification 2025, 2025. URL `https://bimforum.org/wp-content/uploads/2026/01/LOD-Spec-2025-Part-I-Official.pdf`.

[8] O. Disney, M. Roupé, M. Johansson, J. Ris, and P. Höglin. Total BIM on the construction site: a dynamic single source of information. *ITcon*, 28:519–538, 2023. doi:10.36680/j.itcon.2023.027.

[9] C. Tie, S. Sun, J. Zhu, Y. Liu, J. Guo, Y. Hu, H. Chen, J. Chen, R. Wu, and L. Shao. Manual2Skill: Learning to read manuals and acquire robotic skills for furniture assembly using vision-language models, 2025. arXiv:2502.10090.

[10] S. Li, Z. Yan, Z. Wang, and Y. Gao. VLM-MSGraph: Vision language model-enabled multi-hierarchical scene graph for robotic assembly. *Robotics and Computer-Integrated Manufacturing*, 94: 102978, 2025. doi:10.1016/j.rcim.2025.102978.

[11] A. H. Kyaw, R. Gupta, D. Shah, A. Sinha, K. Mathewson, S. Pender, S. Chitta, Y. Koga, F. Ahmed, L. Sass, and R. Davis. Text to robotic assembly of multi component objects using 3D generative ai and vision language models, 2025. arXiv:2511.02162. NeurIPS 2025 (Creative AI Track).

[12] M. He, Y. Zheng, Y. Liu, Z. An, B. Cai, J. Huang, L. Zhou, F. Liu, Z. Li, and L. Zhang. Vision language models cannot plan, but can they formalize?, 2025. arXiv:2509.21576.

[13] H. Oyediran, W. Turner, K. Kim, and M. Barrows. Integration of 4D BIM and robot task planning: Creation and flow of construction-related information for action-level simulation of indoor wall frame installation, 2024. arXiv:2402.03602.

[14] A. Fingrut, S. Altun, and Y. Li. Integration of robotics and computer vision for responsive and autonomous bricklaying in construction. *Technology|Architecture + Design*, 8(2):271–281, 2024. doi:10.1080/24751448.2024.2405406.