Tokenizing Buildings: A Transformer for Layout Synthesis

Manuel Ladron de Guevara
Higharc
manuelrodriguez@higharc.com

Jinmo Rhee
University of Calgary, Canada
jinmo.rhee@ucalgary.ca

Ardavan Bidgoli
Higharc
ardavanbidgoli@higharc.com

Vaidas Razgaitis
Higharc
vaidasrazgaitis@higharc.com

Michael Bergin
Higharc
michaelbergin@higharc.com

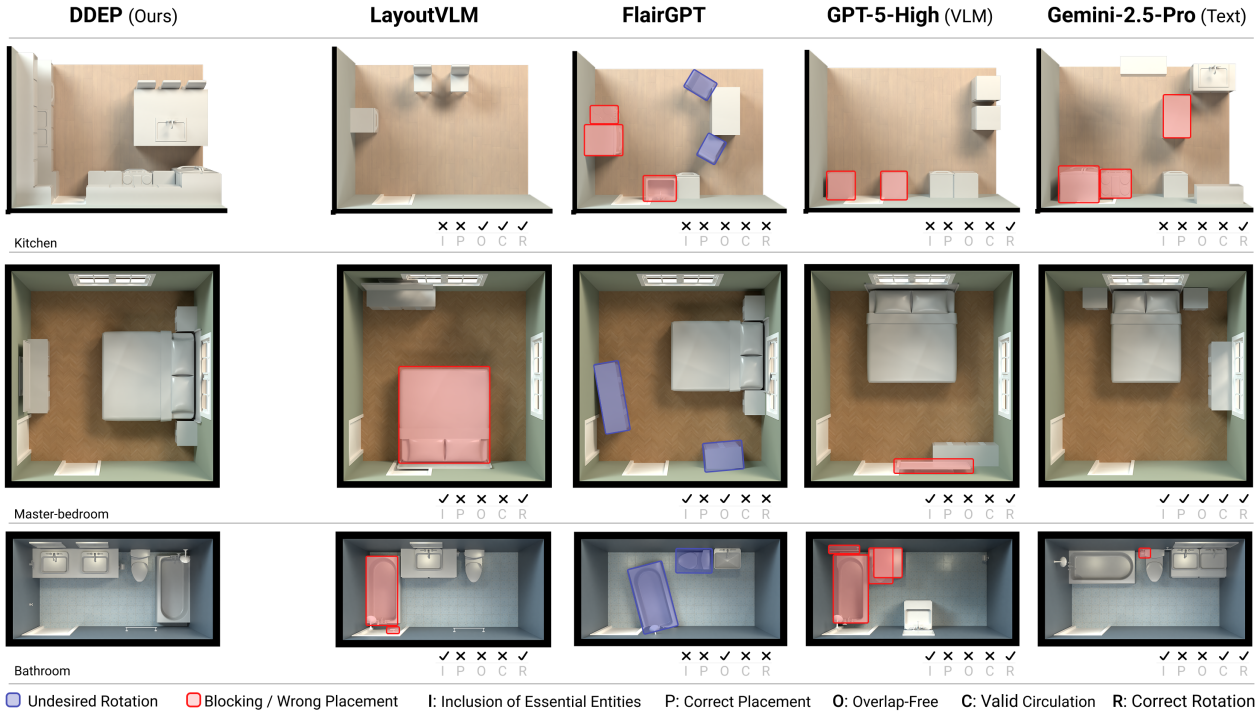

Figure 1. Small Building Model (SBM) is an encoder-decoder Transformer that generates functionally correct and semantically coherent layouts given a room envelope. Each row shows a different room type. Our approach outperforms general-purpose LLMs/VLMs and domain-specific methods.

Abstract

We introduce Small Building Model (SBM), a Transformer-based architecture for layout synthesis in Building Information Modeling (BIM) scenes. We address the question of how to tokenize buildings by unifying heterogeneous feature sets of architectural elements into sequences while preserving compositional structure. Such feature sets are represented as a sparse attribute-feature matrix that captures room properties. We then design a unified embedding module that learns joint representations of categorical and possibly correlated continuous feature groups. Lastly, we train a single Transformer backbone in two modes: an encoder-only pathway that yields high-fidelity room embeddings, and an encoder–decoder pipeline for autoregressive prediction of room entities—referred to as Data-Driven Entity Prediction (DDEP). Experiments across retrieval and generative layout synthesis show that SBM learns compact room embeddings that reliably cluster by type and topology, enabling strong semantic retrieval. In DDEP mode, SBM produces functionally sound layouts—with fewer collisions and boundary violations and improved navigability.

1. Introduction

Professional Computer Aided Design (CAD)/BIM scenes encode rich semantics, hierarchies, and domain constraints, but most strong 3D generators—voxel, mesh, point-cloud, or image-conditioned diffusion models—treat scenes as unstructured geometry [3], yielding visually plausible yet hard-to-edit outputs that frequently violate basic validity rules. Room-scale layout design further demands multi-scale reasoning and parametric outputs compatible with downstream BIM tools, reflecting the highly constrained yet repetitive nature of practice, where designers position walls, doors, casework, and circulation to satisfy codes and programs across hundreds of similar rooms [1, 2, 4, 6]. Decades of computational approaches—from CAD macros and parametric families to rule systems, optimization, and learning-based methods—have aimed to automate this process [15], but progress remains bottlenecked not by algorithms themselves but by the representations they operate on. An effective representation must expose structure, generalize across building types, and maintain BIM-level geometric and semantic coherence, underscoring the need to rethink spatial data encoding as the foundation for scalable layout reasoning.

To this end, we introduce a normalized hierarchical tokenizer that encodes room topology, entity attributes, wall-referenced geometry, and relational structure into compact *BIM-Token Bundles*. We realize these bundles as rows of a sparse attribute–feature matrix and embedded via a mixed-type module that produces a unified token vector for Transformer models. Building on this representation, our *SBM* provides a Transformer backbone for BIM scenes, supporting both encoder-only analysis of room embeddings at scale and encoder–decoder generation through *DDEP*, which autoregressively places room-level entities with mixed categorical and continuous attributes while preserving the compositional structure of BIM data.

In summary, we contribute (1) a normalized hierarchical tokenization of BIM scenes with a mixed-type embedding module, (2) a unified Transformer backbone (SBM) that supports both retrieval and generative layout synthesis through encoder-only and encoder–decoder (DDEP) modes, and (3) a comprehensive evaluation on a professional BIM dataset demonstrating consistently higher scene coherence, geometric validity, and constraint satisfaction than recent LLM/VLM and domain-specific baselines.

2. Related Work

Rule-based and heuristic scene and layout generation. Early work on automatic furniture layout relied on explicit design rules and expert heuristics encoded as constraints. Merrell et al. [14] incorporated interior design guidelines into a cost function and generated layout suggestions by sampling from it. Similarly, Song et al. [20] divided layouts into a few usage modes and applied mode-specific optimization strategies such as recursive subdivision and floor-field methods. Beyond these deterministic rule systems, some approaches convert design guidelines into objective terms and use search-based optimization—such as genetic algorithms—to explore a wider solution space while still operating on the same rule set [9].

Deep learning–based scene and layout generation. Graph-based generative models represent layouts as scene graphs and synthesize object arrangements by predicting node attributes and relational edges [7, 27]. Image- and feature-based neural models—ranging from GANs and transformers [13, 24] to hybrid vectorized systems like HouseDiffusion [19]—learn strong statistical priors for furniture placement yet still depend heavily on training distributions and often require post-hoc refinement to satisfy geometric and functional constraints. Diffusion-based models further improve controllability and semantic consistency, via conditional GAN refinement [16], mixed discrete–continuous diffusion for 3D scenes [8], room-mask-conditioned semantic diffusion [23], multi-view RGB-D generation [17], and text-to-layout diffusion Transformers [18, 21].

LLM/VLM-based scene and layout generation. Language-based models have recently been applied to spatial layout generation by representing scenes as token sequences. LayoutGPT [5] frames layout synthesis as compositional planning, generating object–relation tokens autoregressively, though it remains reliant on prompting and lacks continuous geometric grounding. Vision-language approaches such as LayoutVLM [22] embed spatial cues jointly with textual descriptions to optimize 3D layouts, yet their coupling of vision features and natural language does not yield a normalized geometric token space.

Other systems use LLMs for guided refinement: HouseTune [29] proposes and linguistically adjusts floorplan candidates, while FlairGPT [12] aligns language tokens with furniture and style attributes for interior design exploration. Multi-modal models like LLM4CAD [10] tokenize CAD elements symbolically to support reasoning but operate at the object level without encoding continuous spatial attributes or wall-referenced coordinates. Recently, FloorPlan-DeepSeek [28] formulates floorplan synthesis as autoregressive next-room prediction, encoding each room as a vector of semantic type and geometric attributes. This sequential approach constructs layouts incrementally from partial plans or text prompts, achieving improved structural coherence and enabling downstream editing. However, the approach remains limited to room-level floor-plan arrange-

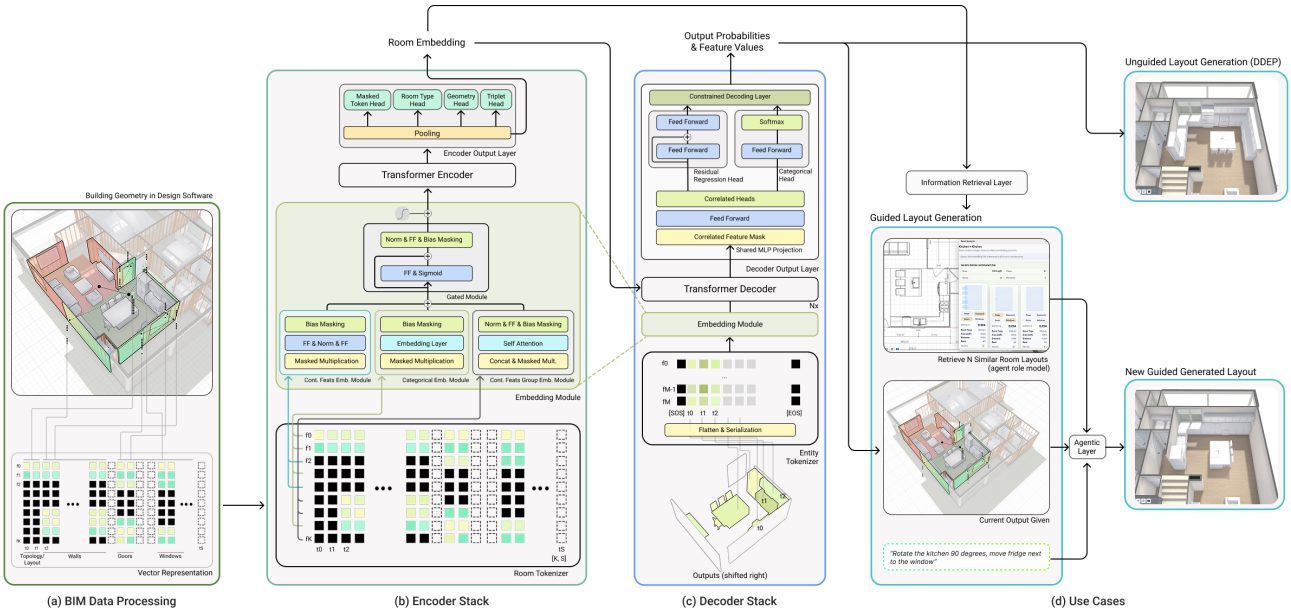

Figure 2. Model overview. (a) BIM data extraction and assembly into a discrete set of token bundles. (b) SBM encoder stack processes the tokenized feature-attribute matrix and outputs a room representation. (c) SBM decoder stack consumes the room representation as memory to the cross-attention layers and the room entities as inputs, trained on next token prediction. (d) Use cases: our SBM is used for three main tasks: DDEP, information retrieval, and user-guided DDEP with the help of an agentic layer.

ment, and does not encode fine-grained BIM elements or wall-hosted entities.

Gaps. Spatial layout research has evolved from rule-based systems to graphs, images, vectorized polygons, and most recently language-based token sequences, each step improving expressiveness. Yet existing representations remain rigid, task-specific, or fragmented, and still fail to unify semantic, relational, and geometric information in a sequence-native form. Most methods yield images, semantic maps, or coarse graphs rather than structured tokens with explicit geometric references and mixed categorical–continuous attributes, limiting constraint-aware reasoning, cross-task generalization, and downstream BIM editability.

This persistent representational gap motivates our approach: a normalized tokenization of BIM scenes coupled with a custom embedding module and a shared Transformer backbone, designed to provide a unified, expressive, and geometrically grounded foundation for scalable spatial layout reasoning.

3. Method

3.1. Problem Setup and BIM Room Representation

We operate on room-level layouts extracted from professional BIM projects (see Sec. 4.1.1 for dataset details). Each room is treated as a parametric layout with semantic attributes, and wall, openings, and furniture/casework entities, all expressed in a local coordinate frame normalized for position and scale.

Envelope vs. contents. We explicitly decompose each room into a *room envelope*—the walls, openings, and coarse layout attributes—and a set of *room contents* (props and casework). We write $r_{\text{env}} = \left(y^{\text{topo}}, y^{\text{layout}}, \mathcal{E}, \mathcal{D}, \mathcal{W}\right), r_{\text{ent}} = \left(\mathcal{P}, \mathcal{C}\right)$ and define the full room as $r = (r_{\text{env}}, r_{\text{ent}})$. Here y^{topo} is a discrete room-type token (e.g., *bedroom*, *kitchen*) and y^{layout} is a layout token that aggregates global scalar attributes such as area and perimeter. The sets $\mathcal{E}, \mathcal{D}, \mathcal{W}$ are walls, doors, and windows, and $\mathcal{P}, \mathcal{C}$ are props and casework, respectively. In our model, the *encoder* operates only on the envelope r_{env}, while the *decoder* operates only on the entity sequence r_{ent} conditioned on the encoder output.

Walls and room topology. The room boundary and internal partitions are represented as a set of N_E directed wall segments $\mathcal{E} = \{e_j\}_{j=1}^{N_E}, e_j = \left(x_j^{(1)}, x_j^{(2)}, a_j^{\text{wall}}\right)$, where $x_j^{(1)}, x_j^{(2)} \in \mathbb{R}^2$ are the endpoints in the local coordinate frame and a_j^{wall} collects wall attributes (e.g., inside/outside thickness, wall condition or construction type, and a discrete edge identifier). The counter-clockwise ordered set $\mathcal{E}$ defines the room polygon and its topology; global geomet-

ric scalars such as area and perimeter are computed from $\mathcal{E}$ and packed into the layout token y^{layout}.

Openings: doors and windows. Doors and windows are modeled as objects attached to specific wall edges with wall-referenced coordinates. For doors we write $\mathcal{D} = \{d_k\}_{k=1}^{N_D}, d_k = \big(j_k, t_k, w_k, a_k^{\text{door}}\big)$, where $j_k \in \{1, \dots, N_E\}$ indexes the supporting edge e_{j_k}, $t_k \in [0,1]$ is a normalized position along that edge, w_k is the opening width, and a_k^{door} includes additional categorical attributes (e.g., door family/type, swing direction). Windows $\mathcal{W} = \{w_\ell\}_{\ell=1}^{N_W}$ are defined analogously with edge attachment, normalized position, width, and window-specific attributes. This wall-referenced parameterization makes openings invariant to absolute translation and scale and directly compatible with BIM editing operations.

Furniture and casework entities. Room contents are represented as two sets of entities: $\mathcal{P} = \{p_u\}_{u=1}^{N_P}, \quad \mathcal{C} = \{c_v\}_{v=1}^{N_C}$, for props (furniture) and casework. Each entity is parameterized by a discrete type and a small set of wall-referenced continuous attributes. For a generic entity $q \in \mathcal{P} \cup \mathcal{C}$ we write $q = \big(c_q, j_q, t_q, \delta_q, s_q, \rho_q, a_q^{\text{ent}}\big)$, where c_q is an entity type identifier (e.g., *bed*, *base_cabinet*), j_q is the supporting edge index, $t_q \in [0,1]$ is the position along that edge, δ_q is a signed lateral offset from the wall into the room, s_q encodes size or footprint parameters (e.g., width/depth), ρ_q is an in-plane rotation angle when applicable (props only), and a_q^{ent} contains any additional categorical attributes. This representation captures both semantic identity and precise placement of entities relative to the room envelope.

Tasks. Given this decomposition, our model supports two tasks. In *encoder-only* mode, the goal is to map the room envelope r_{env} to a compact embedding $z(r_{\text{env}}) \in \mathbb{R}^d$ that preserves semantic and geometric relationships between rooms (for retrieval and analysis). In *encoder–decoder* mode (DDEP), the encoder consumes only the envelope r_{env} to produce a contextual memory, and the decoder autoregressively generates the entity sequence $r_{\text{ent}} = (\mathcal{P}, \mathcal{C})$ as a sequence of tokens (see Fig. 2). The following subsections describe how we convert r_{env} and r_{ent} into BIM-Token Bundles on sparse attribute–feature matrices, embed these mixed-type features, and train a shared Transformer backbone to solve both tasks.

3.2. BIM Tokenization

Given the room envelope r_{env} and contents r_{ent} (Sec. 3.1), we convert each room into two ordered sequences of *BIM-Token Bundles*: an envelope sequence for the encoder and an entity sequence for the decoder. Each bundle corresponds to a single logical element (topology, layout, wall, door, window, or entity) and aggregates all of its semantic and geometric attributes into a sparse feature vector.

Token sequences. On the encoder side, we construct a fixed-order sequence that starts with a special classification token and ends with an end-of-sequence token:

$$\big(\tau^{\text{CLS}}, \tau^{\text{t}}, \tau^{\text{l}}, \tau_1^{\text{e}}, \dots, \tau_{N_E}^{\text{e}}, \tau_1^{\text{d}}, \dots, \tau_{N_D}^{\text{d}}, \tau_1^{\text{w}}, \dots, \tau_{N_W}^{\text{w}}, \tau^{\text{EOS}}\big),$$

followed by padding tokens up to a maximum length S_{enc}. The CLS bundle τ^{CLS} is a learnable summary token used for pooling in encoder-only mode; it does not correspond to any specific BIM element. The topology bundle τ^{t} encodes y^{topo} (room type), the layout bundle τ^{l} encodes y^{layout} (area, perimeter, and other global scalars), each edge bundle τ_j^{e} aggregates the attributes of $e_j \in \mathcal{E}$, and each door or window bundle aggregates the corresponding element in $\mathcal{D}$ or $\mathcal{W}$ with wall-referenced parameters. The EOS bundle τ^{EOS} marks the end of the envelope sequence. This encoder sequence is *envelope-only*: props and casework are not encoded in the encoder input.

On the decoder side, we form a contents-only entity sequence $\big(\tau^{\text{SOS}}, \tau_1^{\text{ent}}, \dots, \tau_T^{\text{ent}}, \tau^{\text{EOS}}\big)$, where each τ_t^{ent} corresponds to one entity $q \in \mathcal{P} \cup \mathcal{C}$ with its type and wall-referenced attributes, and τ^{SOS}, τ^{EOS} are special start/end tokens. The room envelope is provided to the decoder only via cross-attention to the encoder memory, not re-encoded as decoder tokens.

Attribute–feature matrices. We realize these heterogeneous token sequences as *attribute–feature matrices* with a fixed feature axis and variable sequence length. For the encoder, we define $X^{\text{enc}} \in \mathbb{R}^{F_{\text{enc}} \times S_{\text{enc}}}$,

$$X^{\text{enc}} = \begin{bmatrix} x_{1,1}^{\text{enc}} & x_{1,2}^{\text{enc}} & \cdots & x_{1,S_{\text{enc}}}^{\text{enc}} \\ x_{2,1}^{\text{enc}} & x_{2,2}^{\text{enc}} & \cdots & x_{2,S_{\text{enc}}}^{\text{enc}} \\ \vdots & \vdots & \ddots & \vdots \\ x_{F_{\text{enc}},1}^{\text{enc}} & x_{F_{\text{enc}},2}^{\text{enc}} & \cdots & x_{F_{\text{enc}},S_{\text{enc}}}^{\text{enc}} \end{bmatrix},$$

where each *column* $X_{:,s}^{\text{enc}}$ encodes one BIM-Token Bundle at sequence position s, and each *row* corresponds to a specific feature. The first rows are two generic categorical identifiers that are present for every token, a type id to denote the semantic attribute, and a token id to index each semantic instance within each type. The remaining rows collect room-level scalars (e.g., area, perimeter), wall geometry (edge endpoints, lengths, thicknesses), and opening parameters (normalized position along the wall, width, corner distances), as configured in a fixed feature table. Only a subset of these features is active for any given token type; non-applicable entries are filled with a designated padding value, yielding a sparse representation. A complete list of encoder feature rows is provided in the supplement.

We construct the decoder matrix $X^{\text{dec}} \in \mathbb{R}^{F_{\text{dec}} \times S_{\text{dec}}}$ in the same way, now with decoder-specific features such as entity type, edge attachment, wall-referenced coordinates (t_q, δ_q), size parameters s_q, and rotation ρ_q, plus *token_type_id* and *token_id*. Here too, only the rows relevant to each token type are active and the rest are set to $p = -100$. In the next subsection, we describe how these sparse matrices X^{enc} and X^{dec} are mapped to dense token embeddings by a mixed-type embedding module.

3.3. Mixed-Type Embedding Module

The BIM-Token Bundles defined above combine heterogeneous categorical and continuous attributes on a shared feature axis. Our mixed-type embedding module converts the sparse feature matrices X^{enc} and X^{dec} into dense token embeddings suitable for a Transformer, while respecting which features are active for which tokens.

Formally, we define two embedding maps

$$\Phi_{\text{enc}} : \mathbb{R}^{F_{\text{enc}} \times S_{\text{enc}}} \to \mathbb{R}^{S_{\text{enc}} \times d}, \quad \Phi_{\text{dec}} : \mathbb{R}^{F_{\text{dec}} \times S_{\text{dec}}} \to \mathbb{R}^{S_{\text{dec}} \times d},$$

implemented by a shared *FeatureEmbedding* module configured with encoder- or decoder-specific feature definitions. In both cases, each token embedding is constructed by summing the contributions of all its active features.

Feature-wise embeddings and masking. Consider first the encoder. For a single room, the encoder feature matrix X^{enc} has rows $x_f \in \mathbb{R}^{S_{\text{enc}}}$ corresponding to individual features (e.g., *token_type_id*, *area*, *edge_x1*, *t_value*). For each feature f we instantiate a feature-specific embedding function E_f:

- **Categorical features** (e.g., *token_type_id*, *token_id*, edge indices, wall condition) use learnable embedding tables $E_f : \mathbb{Z} \to \mathbb{R}^d$ with a dedicated entry for padding.
- **Scalar continuous features** (e.g., area, perimeter, edge length, relative length) use small multilayer perceptrons $E_f : \mathbb{R} \to \mathbb{R}^d$.
- **Grouped continuous features** (e.g., edge endpoint coordinates, thickness pairs, door/window corner distances) use a *MultiContinuousEmbedding*: a lightweight subnetwork that embeds each scalar in the group, applies self-attention or pooling across the group, and projects the result to $\mathbb{R}^d$.

Let $\mathcal{F}_{\text{enc}}$ be the set of encoder features. For each feature $f \in \mathcal{F}_{\text{enc}}$ and token position s, we define a valid-feature mask $m_{f,s} = \mathbb{1}\big[X^{\text{enc}}_{f,s} \neq -100\big]$, which indicates whether feature f applies to token s. After replacing sentinel values -100 by 0 for numerical stability inside E_f, we compute feature embeddings and apply the mask: $u_{f,s} = E_f\big(X^{\text{enc}}_{f,s}\big) \in \mathbb{R}^d, \tilde{u}_{f,s} = m_{f,s} \cdot u_{f,s}$. This second masking step ensures that inactive features contribute exactly zero, including the bias terms of linear layers. The encoder attention mask further zeroes out all embeddings for padded token positions.

Token embedding as feature sum. The final encoder token embedding at position s is the sum of all active feature embeddings for that token: $e^{\text{enc}}_s = \sum_{f \in \mathcal{F}_{\text{enc}}} \tilde{u}_{f,s} \in \mathbb{R}^d$. Stacking these over sequence positions yields $E^{\text{enc}} = \Phi_{\text{enc}}(X^{\text{enc}}) \in \mathbb{R}^{S_{\text{enc}} \times d}$, optionally augmented with learned positional embeddings. This construction is token-type agnostic: topology, layout, walls, doors, and windows are all embedded via the same mechanism, differing only in which feature rows are active for each token.

Decoder embeddings. The decoder uses the same FeatureEmbedding architecture with a decoder-specific feature set. Given the decoder feature matrix X^{dec}, we instantiate analogous categorical, continuous, and grouped-continuous embedders for features such as *token_type_id*, *token_id*, edge attachments, wall-referenced coordinates (t_q, δ_q), size parameters s_q, and rotations ρ_q. Applying the same masking and summation procedure yields decoder token embeddings $E^{\text{dec}} = \Phi_{\text{dec}}(X^{\text{dec}}) \in \mathbb{R}^{S_{\text{dec}} \times d}$, with the property that entities of different types (props vs. casework, SOS/EOS vs. regular tokens) share the same embedding space but rely on different subsets of features.

By treating each BIM-Token Bundle as a sparse feature column and mapping it through this mixed-type embedding module, we obtain dense, sequence-native token embeddings for both the room envelope and its contents. The next subsection describes how a standard Transformer backbone consumes E^{enc} and E^{dec} in encoder-only and encoder–decoder modes.

3.4. Transformer Backbone and Operating Modes

Given the encoder and decoder token embeddings we employ a standard Transformer encoder–decoder architecture. The same backbone is used in two operating modes: encoder-only (for room embeddings and retrieval) and encoder–decoder (for data-driven entity prediction, DDEP).

Encoder-only: room embeddings and retrieval. In encoder-only mode, we use only the Transformer encoder. Given E^{enc} and an attention mask over non-padding tokens, we compute $M = \text{Enc}_\theta(E^{\text{enc}}) \in \mathbb{R}^{S_{\text{enc}} \times d}$, where Enc_θ is a stack of self-attention and feed-forward blocks. The first position in the sequence corresponds to the special CLS token τ^{CLS} introduced in Sec. 3.2. We obtain a room embedding for the envelope by pooling at this position $z(r_{\text{env}}) = M_0 \in \mathbb{R}^d$, which we use directly for room retrieval and similarity-based analysis. Encoder heads (e.g., room-type classification and masked-token prediction) operate on $z(r_{\text{env}})$ and on the full memory M.

Encoder–decoder: Data-Driven Entity Prediction (DDEP). In DDEP mode, we use the full encoder–decoder Transformer to generate the room contents $r_{\text{ent}} = (\mathcal{P}, \mathcal{C})$ conditioned on the room envelope r_{env}. The encoder processes the envelope as above, producing a memory M that is held fixed during decoding. The decoder takes entity token embeddings E^{dec} and attends both to the prefix of generated tokens (causal self-attention) and to the encoder memory M (cross-attention), $H = \text{Dec}_\theta\big(E^{\text{dec}}, M\big) \in \mathbb{R}^{S_{\text{dec}} \times d}$, where Dec_θ is a stack of masked self-attention, cross-attention, and feed-forward layers. Decoder states in H feed into multi-head prediction layers that produce mixed categorical and continuous outputs for each entity token. We enforce a constrained structured decoding schema which refines the raw predictions.

4. Experiments and Results

We structure this section around our two main parts, DDEP and SBM embeddings, and compare our results to previous approaches. Due to space constraints, we defer training, implementation details and full tokenizer/adapter configurations to the Supplementary Material. In all experiments, we use DDEP in its *unguided* form: the decoder autoregressively generates entity sequences conditioned only on the room envelope and encoder memory, without retrieval-based conditioning or external agent feedback.

4.1. Experimental Setup

4.1.1. Dataset and Setup

We train SBM on a proprietary corpus of residential BIM scenes containing single-family homes with typed rooms (e.g., bathrooms, dining rooms, offices, pantries). Across these constraints, the dataset contains 439,932 training samples over. We construct held-out splits per room type and report all quantitative results on the same test rooms for every method.

Each room is converted to BIM-Token Bundles and then to encoder and decoder sequences as in Sec. 3.2, yielding relatively short sequences compared to text corpora. Over all room types and training regimes, SBM sees on the order of five million tokens in total across encoder and decoder streams, i.e., a few million envelope tokens and a comparable number of entity tokens. Despite this modest scale, the dataset covers a wide range of residential geometries and furniture programs, providing enough diversity to train both the encoder-only embedding pathway and the DDEP decoder for the room types evaluated in Secs. 4.2 and 4.4.

4.1.2. Compared Methods

We evaluate three families of methods on the same held-out rooms. For each room envelope, we construct a serialized text description and, when applicable, a rendered floorplan image. All methods return predicted props and casework, which we map to a unified schema of entity categories and dimensions. For LLM/VLM baselines we use fixed prompts (see Suppl.) and decoding (T=0.7, top-p=0.9 unless stated). VLMs condition on a 1024×1024 floorplan render (orthographic, 20px/m). Free-text outputs are parsed via a rule-based mapper to our inventory. Each method runs 3 seeds; we report mean +/- sd and inference time. For reproducible implementation details, refer to Supplementary Material.

Text LLMs. Claude Sonnet 4.5, Gemini 2.5 Pro, GPT-5, and Qwen3 in text-only mode.

Vision–language models (VLMs). The same base models with image-conditioned variants that receive a floorplan render plus textual context.

Domain-specific methods. LayoutVLM [22], FlairGPT [12], and our DDEP model built on the SBM encoder–decoder backbone.

4.1.3. Layout Quality Metrics

We report three complementary layout metrics: coverage (inventory satisfaction), navigability (functional access), and overlap & clearance (spatial violations). All metrics use the same room geometry, entity dimensions, and inventory specification for every method. Full implementation details are provided in the Supplement.

Coverage. Coverage measures how well a predicted layout $\hat{y}$ satisfies the room-type inventory y, independent of exact placement. Let $\mathcal{I}$ be required items and $\mathcal{G}$ mutually exclusive alternative groups with non-negative weights w_i, w_g:

$$\text{Cov}(y, \hat{y}) = \frac{\sum_{i \in \mathcal{I}} w_i \, s_i(\hat{y}) \; + \; \sum_{g \in \mathcal{G}} w_g \, s_g(\hat{y}) \; - \; p_{\text{extra}}(\hat{y})}{\sum_{i \in \mathcal{I}} w_i \; + \; \sum_{g \in \mathcal{G}} w_g}.$$

Item scores $s_i \in [0, 1]$ handle missing/overfill counts; group scores $s_g \in [0, 1]$ capture satisfaction of any valid alternative; $p_{\text{extra}} \geq 0$ penalizes extraneous categories.

Navigability. The navigability metric evaluates the walkability from door-to-target. Let $\mathcal{P}$ denote all door–target pairs, and let ℓ_t be the shortest collision-free path length for pair t (computed with A*), with $\ell_t^\star$ the straight-line distance inside the room. We define the success rate and detour factor as $\text{SR} = \frac{1}{|\mathcal{P}|} \sum_{t \in \mathcal{P}} \mathbf{1}[\ell_t < \infty]$, $\text{DF} = \frac{1}{|\mathcal{P}_{\text{reach}}|} \sum_{t \in \mathcal{P}_{\text{reach}}} \frac{\ell_t}{\ell_t^\star}$, where $\mathcal{P}_{\text{reach}} = \{t \in \mathcal{P} : \ell_t < \infty\}$. For visualization we also report a scalar navigability score $\text{Nav} = 100 \times \big(\text{SR} - \lambda \, \text{DF}\big)$, with λ chosen on a validation set to balance reachability and detour.

Overlap & clearance. The overlap & clearance metric measures spatial violations using exact polygon geometry.

Method	Coverage ↑ mean ± std	Navigability ↑ mean ± std	Navigability ↑ SR (mean ± std)	Navigability ↓ DF (mean ± std)	OC (%) ↓ mean ± std	Inference Latency (s) ↓ mean ± std
Text-based methods						
Claude Sonnet 4.5	34.1 ± 0.3	2.1 ± 2.7	0.26 ± 0.02	0.69 ± 0.03	33.3 ± 2.5	18.02 ± 12.19
Gemini 2.5 Pro	67.1 ± 1.6	58.0 ± 8.3	0.69 ± 0.07	0.30 ± 0.03	10.2 ± 1.6	42.51 ± 13.25
GPT-5 Medium	56.8 ± 1.5	51.2 ± 8.8	0.63 ± 0.07	0.32 ± 0.04	12.9 ± 1.8	100.86 ± 65.58
Qwen3	60.5 ± 1.4	34.0 ± 0.8	0.50 ± 0.01	0.45 ± 0.02	18.1 ± 1.6	21.27 ± 13.32
VLM methods						
Claude Sonnet 4.5	44.9 ± 3.4	52.7 ± 1.7	0.64 ± 0.01	0.32 ± 0.03	23.7 ± 2.0	18.71 ± 3.62
Gemini 2.5 Pro	73.2 ± 1.2	54.6 ± 2.2	0.67 ± 0.02	0.34 ± 0.02	11.9 ± 1.9	43.16 ± 13.16
GPT-5 Medium	55.5 ± 1.4	54.3 ± 10.4	0.65 ± 0.08	0.29 ± 0.05	12.3 ± 3.9	164.43 ± 105.88
GPT-5 High	58.3 ± 0.0	57.8 ± 54.0	0.67 ± 0.42	0.27 ± 0.39	12.1 ± 25.5	124.18 ± 78.34
Qwen3	75.9 ± 1.5	48.7 ± 1.0	0.60 ± 0.02	0.33 ± 0.02	12.4 ± 1.6	439.04 ± 377.43
Domain-specific methods						
LayoutVLM	–	40.3 ± 59.8	0.55 ± 0.45	0.41 ± 0.45	4.9 ± 11.4	144.99 ± 36.50
FlairGPT	46.6 ± 36.5	28.2 ± 61.0	0.46 ± 0.46	0.51 ± 0.46	5.0 ± 12.9	1133.61 ± 1596.40
DDEP (Ours)	**98.2 ± 6.3**	**82.4 ± 40.7**	**0.89 ± 0.31**	**0.18 ± 0.29**	**3.2 ± 10.2**	**3.18 ± 0.18**

Table 1. Comparison on the held-out set. We report mean ± std. Higher is better for Coverage, Navigability (mean and SR); lower is better for OC and DF. Best overall (SOTA) also in **bold**. LayoutVLM requires an initial furniture list as input, so it is excluded from Coverage metric. Latency is reported as mean ± std per room, measured end-to-end on an AWS EC2 `g5.xlarge` instance.

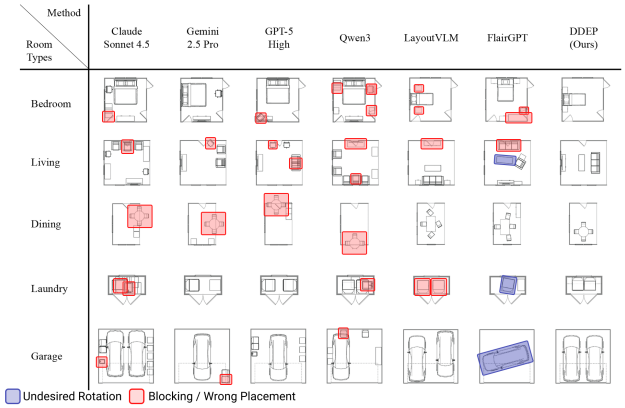

Figure 3. Qualitative comparison of generated layouts across five room types, showing representative results from seven baseline methods and our DDEP model.

We compute four normalized terms: entity overlap fraction (EOF), global overlap area (GOA), door clearance intrusion (DCI), and wall-bounds violation (WBV). These are combined into a single overlap/clearance score

$$\mathrm{OC} = w_{\mathrm{EOF}}\,\mathrm{EOF} + w_{\mathrm{GOA}}\,\mathrm{GOA} + w_{\mathrm{DCI}}\,\mathrm{DCI} + w_{\mathrm{WBV}}\,\mathrm{WBV},$$

with fixed weights $w_\cdot$ shared for all methods.

4.2. DDEP Quantitative Comparisons

On the held-out benchmark (Section 4), we compare layout generators using three metrics. Text LLMs achieve moderate coverage but weak functional quality (under-furnishing, low reachability, higher OC). VLMs improve some axes but exhibit clear trade-offs (gains in coverage often accompany reduced reachability or more collisions). Domain-specific baselines (LayoutVLM, FlairGPT) show lower violation rates yet remain weak on both coverage and navigability under our unified schema. DDEP occupies a distinct regime: near-complete coverage, the highest navigability, and the lowest OC—despite placing substantially more entities than any baseline. No competing method attains high coverage and high navigability simultaneously without incurring larger spatial errors, positioning DDEP as state of the art on this benchmark.

4.3. Qualitative Comparisons

Figure 3 shows example layouts for several room types from DDEP, VLMs, and domain-specific methods. VLM baselines often overfill rooms with extra casework and furniture, blocking circulation and door clearances, especially in living rooms and garages. Domain-specific methods tend to place more appropriate furniture types but still leave narrow or blocked paths around key elements. In contrast, DDEP produces layouts that satisfy the room program while keeping clear, door-connected circulation bands and respecting wall-referenced anchoring, visually matching the trends reported in our quantitative metrics.

4.4. Embedding Evaluation and Encoder Ablations

We evaluate SBM's encoder-only pathway for room embedding quality through retrieval and clustering benchmarks. As an ablation study, we compare against state-of-the-art

Method	nDCG	Triplet	Entity Sim	Clust.(NMI)
SBM	62.6	**100.0**	**78.9**	**0.640**
E5-Large-v2	**86.7**	99.9	78.4	0.371
BGE-Large-en-v1.5	80.1	99.9	77.0	0.358
GTE-Large-en-v1.5	74.3	99.8	77.0	0.335

Table 2. Type-Constrained Retrieval Performance @ k=5

Method	NMI	ARI	Silhouette	Spearman
SBM	**0.640**	**0.307**	**0.152**	0.145
E5-Large-v2	0.371	-0.019	0.051	**0.315**
BGE-Large-en-v1.5	0.358	-0.003	0.069	0.235
GTE-Large-en-v1.5	0.335	-0.002	0.067	0.133

Table 3. Clustering and Geometric Correlation Metrics

text embedding baselines to demonstrate SBM's encoder capacity and highlight its complementary capabilities for geometric and spatial understanding. We compare against *E5-Large-v2* [25]: 560M parameters, pre-trained on Common Crawl and Wikipedia, *BGE-Large-en-v1.5* [26]: 335M parameters, pre-trained on web-scale text data, and *GTE-Large-en-v1.5* [11]: 335M parameters, pre-trained on web-scale text corpora. These baselines are pre-trained on billions of text tokens, representing orders of magnitude more training data than SBM's domain-specific dataset. For fair comparison, we serialize room data to text format, enabling text embeddings to process the same room information.

Evaluation Metrics We assess retrieval quality with nDCG@k (graded relevance) and Success@k/MRR/Precision@k, structural consistency with type-constrained nDCG@k and triplet accuracy, entity-level similarity via overlap of props/casework/doors, and global organization with clustering metrics (NMI/ARI/Silhouette) and geometric rank correlation (Spearman/Kendall) against a geometry oracle. Full metric definitions and cross-type retrieval tables are provided in the Supplement; here we focus on type-constrained retrieval and clustering/correlation.

Table 2 shows that text encoders lead on ranking quality within type (nDCG: E5 86.7, BGE 80.1, GTE 74.3 vs. SBM 62.6), while SBM excels at structure: it attains perfect triplet accuracy (100.0), the highest entity overlap (78.9), and stronger type organization (NMI 0.640). In short, text models rank better among same-type rooms, whereas SBM preserves within-type geometry and entities more faithfully.

As summarized in Table 3, SBM yields markedly better global organization (NMI/ARI/Silhouette: 0.640/0.307/0.152) than text baselines (NMI 0.335–0.371), but aligns less with our geometry oracle in rank correlation (Spearman 0.145 vs. E5's 0.315). This indicates a tighter type-structured space for SBM (Fig. 4) but stronger oracle-ranked agreement for E5.

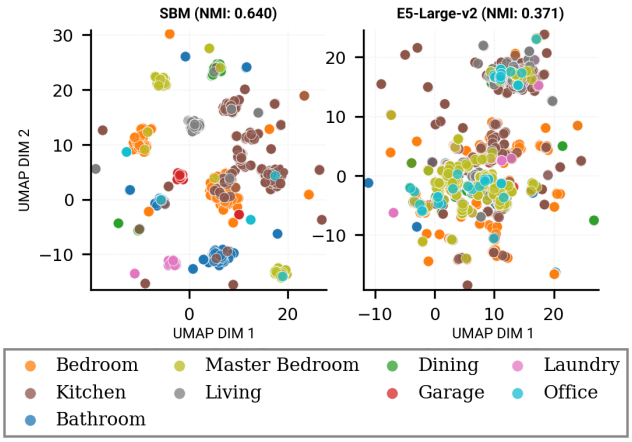

Figure 4. UMAP visualization of room embeddings colored by room type category. SBM embeddings (left, NMI: 0.640) exhibit well-separated clusters with distinct boundaries between room types, demonstrating superior geometric and spatial understanding. E5-Large-v2 embeddings (right, NMI: 0.371) show more intermingled clusters with blurred boundaries, indicating weaker room type separation. The 1.7× higher NMI score for LBM reflects its specialization in capturing geometric structure and spatial relationships inherent in building layouts, rather than semantic similarity alone.

5. Conclusion

We introduced the SBM, a Transformer architecture tailored to BIM-like room layouts through BIM-Token Bundles, sparse attribute–feature matrices, and a mixed-type embedding module shared by encoder and decoder. This representation treats topology, walls, openings, and entities as sequence-native tokens with joint categorical and continuous attributes, enabling a single backbone to support both encoder-only room embeddings and an encoder–decoder pipeline (DDEP). On a BIM-grounded benchmark with specification-aware coverage, navigability, and overlap/clearance metrics, DDEP delivers near-complete inventory satisfaction, state-of-the-art navigability, and the lowest violation rates across LLM/VLM and domain-specific baselines. Encoder ablations show that SBM yields geometry- and entity-aware embeddings that complement generic text encoders, trading some raw ranking strength for substantially better within-type structure. Our study suggests that modestly sized, domain-specific sequence models over well-designed BIM tokenizations can outperform much larger general-purpose systems on layout quality.

Limitations. Advanced vertical constraints like sloped ceilings or stairs/elevation changes, and MEP/service routing are not modeled. The benchmark largely comprises residential rooms from our corpus, so distributional shifts (styles, scales, local codes) may affect external validity.

References

[1] Mario Carpo. *The Alphabet and the Algorithm*. The MIT Press, 1st edition, 2011. 2

[2] Mario Carpo. *The second digital turn: design beyond intelligence*. MIT press, 2017. 2

[3] Charles N. Eastman. *Spatial synthesis in computer-aided building design*. Elsevier Science Inc., 1975. 2

[4] Charles N. Eastman. The Use of Computers Instead of Drawings in Building Design. *AIA Journal*, 63, 1975. 2

[5] Weixi Feng, Wanrong Zhu, Tsu-Jui Fu, Varun Jampani, Arjun R. Akula, Xuehai He, Sugato Basu, Xin Eric Wang, and William Yang Wang. Layoutgpt: Compositional visual planning and generation with large language models. In *Advances in Neural Information Processing Systems*, 2023. 2

[6] Forest Flager and John Riker Haymaker. A comparison of multidisciplinary design, analysis and optimization processes in the building construction and aerospace industries. 2007. 2

[7] Ruizhen Hu, Zeyu Huang, Yuhan Tang, Oliver van Kaick, Hao Zhang, and Hui Huang. Graph2plan: Learning floorplan generation from layout graphs. *arXiv preprint arXiv:2004.13204*, 2020. 2

[8] Song Hu et al. MiDiffusion: Mixed diffusion for 3d indoor scene synthesis. *arXiv preprint arXiv:2405.21066*, 2024. 2

[9] Peter Kán and Hannes Kaufmann. Automated interior design using a genetic algorithm. In *Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology*, pages 1–10, New York, NY, USA, 2017. Association for Computing Machinery. 2

[10] Xingang Li, Yuewan Sun, and Zhenghui Sha. Llm4cad: Multi-Modal large language models for three-dimensional computer-aided design generation. In *Proceedings of the ASME 2024 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE 2024)*, page V006T06A015. ASME, 2024. 2

[11] Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*, 2023. 8

[12] Gabrielle Littlefair, Niladri Shekhar Dutt, and Niloy J. Mitra. Flairgpt: Repurposing llms for interior designs, 2025. EUROGRAPHICS 2025. 2, 6

[13] Yang Liu and Guanjie Wang. Exploration of the Indoor Layout Optimization Model in Computer-Aided Visual Analysis. *Computer-Aided Design and Applications*, pages 167–180, 2024. 2

[14] Paul Merrell, Eric Schkufza, Zeyang Li, Maneesh Agrawala, and Vladlen Koltun. Interactive furniture layout using interior design guidelines. *ACM Trans. Graph.*, 30(4):87:1–87:10, 2011. 2

[15] Javier Monedero. Parametric design: a review and some experiences. *Automation in Construction*, 9(4):369–377, 2000. 2

[16] Nelson Nauata, Wei-Chiu Ma Chang, Yasutaka Furukawa, and et al. House-gan++: Generative adversarial layout refinement network towards intelligent computational agent. In *CVPR*, 2021. 2

[17] Hieu T. Nguyen, Yiwen Chen, Vikram Voleti, Varun Jampani, and Huaizu Jiang. Housecrafter: Lifting floorplans to 3d scenes with 2d diffusion model. In *arXiv preprint arXiv:2406.20077*, 2024. 2

[18] X. Ran et al. Directlayout: Direct numerical layout generation for 3d indoor scene synthesis. *arXiv preprint arXiv:2506.05341*, 2025. 2

[19] Mohammad Amin Shabani, Sepidehsadat Hosseini, and Yasutaka Furukawa. Housediffusion: Vector floorplan generation via a diffusion model with discrete and continuous denoising, 2022. 2

[20] Peihua Song, Youyi Zheng, Jinyuan Jia, and Yan Gao. Web3D-based automatic furniture layout system using recursive case-based reasoning and floor field. *Multimedia Tools and Applications*, 78(4):5051–5079, 2019. 2

[21] D. Srivastava et al. Lay-your-scene: Natural scene layout generation with diffusion transformers. *arXiv preprint arXiv:2505.04718*, 2025. 2

[22] Fan-Yun Sun, Weiyu Liu, Siyi Gu, Dylan Lim, Goutam Bhat, Federico Tombari, Manling Li, Nick Haber, and Jiajun Wu. Layoutvlm: Differentiable optimization of 3d layout via vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 29469–29478, 2025. 2, 6

[23] X. Sun et al. SemLayoutDiff: Semantic layout generation with diffusion models. *arXiv preprint arXiv:2508.18597*, 2025. 2

[24] Hanan Tanasra, Tamar Rott Shaham, Tomer Michaeli, Guy Austern, and Shany Barath. Automation in Interior Space Planning: Utilizing Conditional Generative Adversarial Network Models to Create Furniture Layouts. *Buildings*, 13(7): 1793, 2023. Publisher: Multidisciplinary Digital Publishing Institute. 2

[25] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pretraining. *arXiv preprint arXiv:2212.03533*, 2022. 8

[26] Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. C-pack: Packaged resources to advance general chinese embedding, 2023. 8

[27] Kun Xu, Lingfei Wu, Zhiguo Wang, Yansong Feng, Michael Witbrock, and Vadim Sheinin. Graph2seq: Graph to sequence learning with attention-based neural networks, 2018. 2

[28] Jun Yin, Pengyu Zeng, Jing Zhong, Peilin Li, Miao Zhang, Ran Luo, and Shuai Lu. Floorplan-deepseek (fpds): A multimodal approach to floorplan generation using vector-based next room prediction. *arXiv preprint*, arXiv:2506.21562, 2025. 2

[29] Ziyang Zong, Guanying Chen, Zhaohuan Zhan, Fengcheng Yu, and Guang Tan. Housetune: Two-stage floorplan generation with LLM assistance, 2024. 2