Title

SIM 2.0: Floor Plan Vectorization via VLM Semantic Normalization with Pixel-Wise Self-Consistency

Authors

Halleluyah, Brhanemesqel

Manduchi, Roberto

Publication Date

2026-11-02

SIM 2.0: Floor Plan Vectorization via VLM Semantic Normalization with Pixel-Wise Self-Consistency

Halleluyah Brhanemesqel
University of California, Santa Cruz
Santa Cruz, CA, USA
hbrhanem@ucsc.edu

Roberto Manduchi
University of California, Santa Cruz
Santa Cruz, CA, USA
manduchi@soe.ucsc.edu

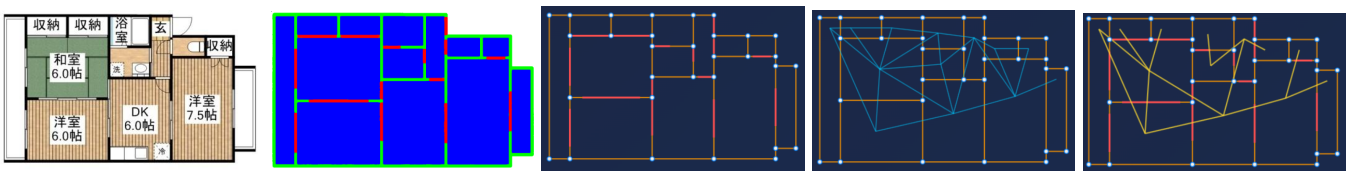


Figure 1: The SIM 2.0 pipeline at a glance. From left to right: input floor plan, RGB consensus mask from N=24 Gemini samples, extracted vertices and edges (orange: edges, red: doors/openings), room connectivity graph, and traversability graph.

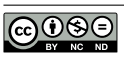
Abstract

This contribution addresses the challenge of converting rasterized architectural floor plans into structured, vectorized representations suitable for spatial reasoning tasks. Existing approaches either rely heavily on manual input or use machine learning models that generalize poorly across diverse graphical styles, limiting scalability and robustness. To overcome these limitations, we introduce SIM 2.0, a generative AI-based system that enables automatic floor plan vectorization across a wide range of styles without retraining.

SIM 2.0 adopts a two-stage pipeline in which a vision-language model first transforms input floor plans into a simplified, standardized representation, followed by deterministic image processing techniques to extract geometric and semantic structures. To mitigate inconsistencies and artifacts inherent in generative models, the system incorporates a pixel-level self-consistency mechanism that aggregates multiple stochastic outputs via majority voting. Experimental results demonstrate that SIM 2.0 outperforms state-of-the-art methods on both in-domain and out-of-domain styles, producing accurate and reliable vectorizations where prior approaches fail. This work highlights the potential of combining generative AI with classical algorithms to achieve robust, scalable solutions for structured spatial data extraction.

CCS Concepts

• **Computing methodologies** → **Image segmentation**; *Shape analysis*; • **Information systems** → *Geographic information systems.*



SIGSPATIAL '26, Riverside, CA, USA

ACM ISBN 979-8-4007-2950-8/2026/11
https://doi.org/10.1145/3841645.3844189

Keywords

Floor plan vectorization, semantic segmentation, wall skeleton extraction, graph-based room detection, RGB semantic mask, vision–language models, self-consistency, Gemini style transfer

ACM Reference Format:
Halleluyah Brhanemesqel and Roberto Manduchi. 2026. SIM 2.0: Floor Plan Vectorization via VLM Semantic Normalization with Pixel-Wise Self-Consistency. In *The 34th ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL '26), November 03–06, 2026, Riverside, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3841645.3844189

1 Introduction

The vectorized representation of architectural floor plans is a prerequisite for a wide range of spatial reasoning tasks, including indoor navigation, building information modeling, and accessibility analysis. By "vectorized", we mean representations in terms of geometric primitives (lines, areas, points), their joint structure (e.g. a traversability graph), and associated semantics (e.g., the name and type of a room). Although all modern building maps are born digital, the original CAD floor plans are seldom available to the general public. When maps are available online, they are normally in the form of "raster" image files, rendered in various styles (JPEG, PNG) or embedded in PDF documents. While generally fulfilling the purpose of helping a viewer find desired locations (e.g., shops in a mall, the elevator or the bathroom in a hospital) and determining the best route there, image files are of little use when a vectorized form of the floor plan is needed. The problem is particularly acute for older buildings, which constitute a substantial proportion of the Western building stock: structures erected before the era of digital drafting often exist only as paper drawings, hand-traced blueprints, or low-resolution scans. For these buildings, a raster image is frequently the sole available representation of the floor plan, making automated vectorization not merely convenient but the only scalable path to structured spatial data.

Several tools have been created to enable "vectorizing" a floor plan image. For example SIM[1] (*Semantic Interior Mapology* [16, 17]) is a web app that allows one to very quickly "trace" an existing floor plan and transform it into GeoJSON or IMDF formats. Using SIM, one selects a particular wall orientation, and clicks on all walls in the building with that orientation. After repeating this for all wall orientations (typically no more than two or three in most buildings), all potential wall corners are automatically identified. The user then defines each space (be it a room, a hallway or a larger space) by clicking on the wall corners defining this space, identifying whether two nearby corners are joined by a wall (vs. open space), and marking any entrances (doors through the walls.)

SIM and similar methods, however, still require human manual input, and therefore scale poorly with large and complex building layouts. Automatic map vectorization would thus be very desirable, and many research projects have addressed this task. Unfortunately, the diversity of graphical conventions, varying line weights, annotation styles, hatching patterns, and scan artifacts, poses a significant challenge for automated vectorization. Existing machine learning methods (e.g., [9, 14]) have been shown to successfully vectorize floor plans in the style they have been trained for, but fail spectacularly when deployed on different style maps (see examples in Sec. 4.7). Retraining these models on a new map style requires obtaining a large number of labeled floor plans in that style, making this approach poorly scalable and ineffective.

This paper introduces **SIM 2.0**, a generative AI-based system for automatic floor plan vectorization that works on a wide variety of graphical map styles, without the need for retraining. In our experiments, SIM 2.0 produces more accurate results than state-of-the-art algorithms (Raster-to-Vector [14] and Raster-to-Graph [9]) on the map style on which these algorithms were trained. It gives similarly good results when tested on other styles, where these two systems are unable to produce any meaningful output.

SIM 2.0 relies on a foundational AI model (we use `gemini-3-pro-image-preview` for this work), which has been trained on extremely large sets of examples. Such models are naturally multimodal, meaning that they can seamlessly solve tasks that involve spatial reasoning and language, among other modalities. One could thus be tempted to solve the vectorization problem in a single shot: prompt the system to generate a structured textual description (e.g., in JSON format) of the map represented in the raster image. As we show experimentally in Sec. 4.5, the single-shot approach produces unsatisfactory results, even for relatively simple floor plans. In contrast to the naive single-shot scheme, SIM 2.0 is structured in a sequence of two layers, the first of which uses generative AI. This approach was inspired by the two-stage algorithm of Kim et al. [12], in which the first stage (*normalization*) first transforms the floor plan image into a similar image but with a standardized style, while the second stage vectorizes this image using a vectorizer derived from Raster-to-Vector [14].

Our design concept for SIM 2.0 was prompted by two observations: (1) the normalization stage of [12], which is based on a popular style transfer algorithm (pix2pix [10]), still requires re-training when a new graphic style is used for the input floor plan; (2) by choosing an appropriately simple format for the normalized image, vectorization can be performed using simple classic image processing algorithms. In particular, SIM 2.0 uses a simple convention for the intermediate normalized image: walls and doors are drawn as simple lines with different colors, and spaces (rooms, hallways or corridors) are filled with a third color. Areas outside the building are colored in a fourth color. We found that, using a simple exemplar prompt, Gemini is able to consistently produce accurate results for different input styles, *without the need for retraining*. A sequence of deterministic pixel-, edge- and area-level simple algorithms (thresholding, connected components, gap closing, skeletonization, and ray probing) are then used for vectorization, starting from the normalized image.

Like all vision-language models (VLMs), Gemini is prone to inconsistent or noisy artifacts (mis-colored pixels, hallucinated rooms, fragmented walls) that corrupt downstream geometry. In order to increase the reliability of style transfer (from input to standardized style), we employ the *self-consistency* principle of Wang et al. [18], that leverages the stochastic nature of VLMs. Specifically, we obtain several versions of the intermediate normalized image by independently running the VLM multiple times, using the same prompt. Each pixel of these images may have one of four colors, representing the semantic classes considered (wall, door, internal spaces, outer area), but the color may differ across the several generated versions. We then generate one normalized image by majority voting for each pixel across all versions. Intuitively, we assume that stochastic errors disagree across versions, while correct structure repeats, so majority voting may dampen hallucinations and produce a single mask substantially more reliable than any individual version. To our knowledge, this is the first application of Wang et al.'s [18] self-consistency aggregation at the pixel level for VLM-based structured image generation.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 describes the SIM 2.0 pipeline: semantic normalization with self-consistency (Sec. 3.1), the deterministic vectorization stages (Sec. 3.2), and the JSON output (Sec. 3.3). Section 4 reports quantitative results against Raster-to-Vector and Raster-to-Graph baselines, and Sec. 4.7 examines two failure cases that expose distributional and architectural limitations of those baselines. Section 5 concludes.

2 Related Work

Existing approaches to floor plan vectorization fall into three categories: methods based on low-level pixel manipulation and heuristic geometry, methods that employ deep learning for semantic extraction (typically followed by pixel-level post-processing), and manual tracing approaches.

Low-level pixel manipulation approaches. Early work relied on classical image processing to detect structural elements directly from raster images. Macé et al. [15] employed the Hough transform combined with image vectorization for wall and door detection, using recursive decomposition under the assumption of roughly convex room shapes. Ahmed et al. [2] detected and labeled rooms by combining a structural preprocessing stage that differentiates thick, medium, and thin lines [1] with OCR on the text layer for room labeling. De las Heras et al. [4] proposed a notation-invariant wall detector using unsupervised run-length analysis, enabling wall

[1] https://sim.acad.ucsc.edu

detection across diverse drawing conventions without notation-specific annotated training data. Gimenez et al. [7] recognized walls by detecting parallel lines separated by a fixed distance, and reconstructed 3D building models from scanned 2D floor plans. These methods require no training data but depend on geometric assumptions (parallel lines, fixed wall thickness, convex rooms) that limit generalization across drawing styles.

Deep learning approaches. More recent methods train neural networks for pixel-wise semantic segmentation of floor-plan elements, then apply geometric post-processing to extract vector structure. Dodge et al. [6] trained a fully convolutional network for wall segmentation. Liu et al.'s **Raster-to-Vector** [14] uses a neural network to predict junctions and integer programming to assemble them into vector primitives. Kalervo et al. [11] presented an improved multi-task CNN that builds on the Raster-to-Vector architecture. Hu et al. [9] introduced **Raster-to-Graph**, which frames floor plan recognition as a structural graph prediction problem using an autoregressive visual-attention Transformer to iteratively predict wall junctions and segments in graph-traversal order. While Raster-to-Graph achieves co-recognition of structure and semantics through a single network, it operates only on 512×512 inputs and shares the fundamental limitation of all supervised approaches: large-scale annotated training data is required, and the model must be retrained for new domains.

Gaps addressed by SIM 2.0. Two gaps follow from the survey above. The first is a *training-data gap*. Modern deep-learning pipelines are bounded by the corpus they were trained on: a model trained on Raster-to-Vector style residential plans performs well on inputs that resemble that corpus and degrades on anything outside it. Extending such a system to a new domain; hospital blueprints, university campus plans, pre-digital archival scans; is not a matter of inference-time configuration but of collecting and annotating a fresh training set, which for many domains of interest does not exist and cannot be assembled at reasonable cost. Heuristic methods are bounded in a different way: they require no training data, but their geometric assumptions (parallel walls at fixed thickness, roughly convex rooms) hard-code one drawing convention and fail on others. The second is a *representation gap*. Geometric vectorization pipelines (e.g., Raster-to-Vector [14], Raster-to-Graph [9]) terminate at wall vectors, room polygons, and at best a list of door lines: they recover precise geometry but do not record which two rooms a given door connects. Conversely, systems that *do* extract traversable connectivity from floor-plan images produce only a topological abstraction. Tesseract [3], for example, detects and classifies each door by the regions it separates to emit a navigable graph. However, its rooms collapse to labeled nodes, no wall geometry, vertices, or room polygons are emitted, and the pipeline depends on printed room-label text to seed its segmentation. Downstream consumers (robot navigation, egress simulation, accessibility auditing) need both: precise geometry to reason about spatial extent, and a traversability relation to distinguish room adjacency (rooms sharing a wall) from traversable connectivity (rooms reachable through a door). To our knowledge, SIM 2.0 is the first system to deliver both in a single, training-free output that requires no textual labels.

SIM 2.0 closes both gaps directly. To handle the training-data gap, we invoke a vision language model (Gemini) at inference time as a one-shot semantic normalizer that maps the input to a small fixed RGB palette of structural classes (walls, doors, rooms, exterior) before any geometric extraction occurs. Because Gemini is a general-purpose foundation model pretrained on web-scale image–text data, it has already seen floor plans across the full range of architectural styles, draughting conventions, and image qualities that appear on the public internet, and we access this coverage through style-transfer prompting rather than supervised fine-tuning. The pipeline therefore accepts floor-plan inputs of arbitrary resolution and style without a domain-specific training set and without retraining when applied to a new domain. To close the representation gap, the deterministic vectorization stage recovers precise geometry (vertices, wall edges, and room polygons) and, over those rooms, builds two relational graphs. The first is a *room-connectivity graph* whose edges are shared wall segments between neighbouring rooms; the second is a *traversability graph* whose edges are doors. Room adjacency and traversable connectivity are thus represented as distinct, first-class structures. This allows routing and connectivity queries to reduce to standard graph traversals, while precise geometry-dependent tasks (e.g., validating wheelchair clearances, calculating line-of-sight, or simulating crowd flow against physical walls) read directly off the geometric attributes attached to each node and edge without requiring post-hoc reconstruction.

3 The SIM 2.0 Pipeline

The pipeline consists of four stages: (A) semantic normalization with pixel-wise self-consistency; (B) color separation and skeletonization; (C) graph extraction (vertices, edges, rooms, edge–room matching, door mapping); and (D) structured JSON output.

3.1 Stage A: Semantic Normalization with Self-Consistency

RGB color contract. The interface between the semantic-interpretation layer and the deterministic geometry layer is a four-class RGB mask whose color encoding is fixed. Specifically: Room interiors are drawn in blue; Structural walls in green; Door openings in red; Exterior areas in white.

One-shot style-transfer prompting. The normalization is performed via Gemini vision-based style transfer (Fig. 2). Specifically, the Gemini VLM is provided with the input floor plan alongside a reference RGB mask as a style exemplar, as well as with a short prompt that asks the model to preserve the input's geometry while adopting the colors and door conventions of the exemplar. This intentionally separates semantic interpretation (identifying structural components) from geometry extraction (precisely defining each element's extent and boundaries). The downstream vectorization module then operates on this simple colored image, which greatly simplifies its task.

Pixel-wise self-consistency aggregation. Any VLM such as Gemini has a stochastic component, meaning that, if run multiple times with the same input, it will generate different results. As shown by Wang et al. in the context of text reasoning [18], VLM's stochastic

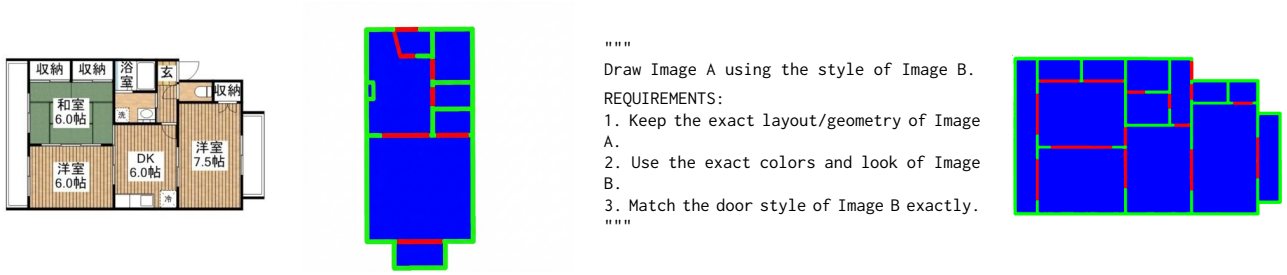


Figure 2: Semantic normalization via Gemini style transfer. Left: input floor plan. Center-left: style reference mask. Center-right: text prompt. Right: generated RGB semantic mask. The style transfer collapses diverse drawing conventions into the four-class palette

behavior can be leveraged (at the cost of increased computation) for increased accuracy, by simply aggregating these multiple outputs (from same input) via majority vote. While this "self-consistency" strategy is simple to implement for categorical tasks (e.g., predicting the class of an object in an image), porting it to the case of style transfer (which generates a whole new image) is not straightforward. We propose a simple approach, that leverages the fact that pixels in the generated image can only take one of four values. Specifically, we query the VLM N times with the same input, obtaining N colored style transferred images. Then, we generate a final image (*consensus mask*), each pixel of which is assigned the color that was most represented (at the same pixel location) across the N images. Intuitively, stochastic errors that vary across the N samples are suppressed, while correct structural elements that recur across samples are reinforced. In the experiments described in this paper we use N=24. Sec. 4.6 shows how increasing the number N of samples leads to increased vectorization accuracy. The parameters for the Gemini VLM are: sampling temperature 1.0, top-p 0.95.

3.2 Stages B-C: Deterministic Vectorization

Once the consensus mask is obtained, a (deterministic) pipeline extracts a structured graph through nine numbered steps organized into three phases: *Color Separation*, *Geometry* (skeletonization, vertex and edge extraction), and *Topology* (room detection, edge–room matching, door mapping). Note that all constant values expressed in pixel units in the pseudo-code are rescaled based on the input image resolution to ensure consistent behavior across different input sizes.

Step 1: HSV color separation. The RGB mask is converted to HSV space for robust color detection. Blue room regions are isolated by hue $\in [0.45, 0.75]$ with saturation > 0.2; green wall regions by hue $\in [0.20, 0.45]$; red door regions, whose hue wraps around 0/1, by hue < 0.08 or > 0.92 with saturation > 0.3 (Fig. 3).

Step 2: Wall gap closure. Doors introduce discontinuities in the wall network: if walls are skeletonized directly (see Step 3 below), the wall graph fragments into disconnected segments at every door opening. We address this by temporarily bridging door gaps prior to skeletonization. The binary door mask is dilated by few iterations of isotropic morphological dilation, and the dilated door pixels are unioned with the wall mask. This produces a topologically closed

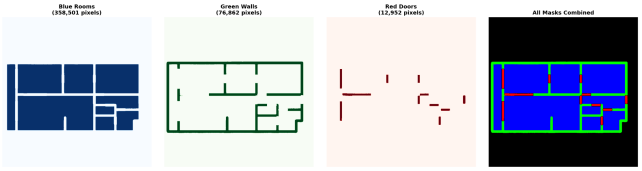


Figure 3: HSV thresholding isolates the three semantic classes: blue rooms, green walls, and red doors.

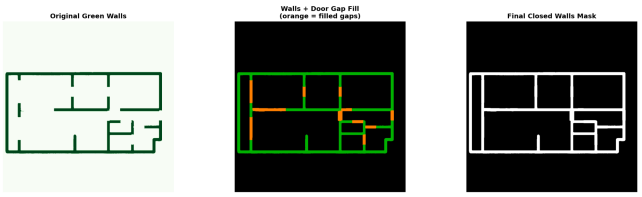


Figure 4: Wall gap closure via door dilation. Door gaps are bridged so wall connectivity is topologically closed prior to skeletonization.

wall network suitable for skeletonization (Fig. 4). The original (undilated) door mask is retained separately for door to edge mapping in Step 9.

Step 3: Skeletonization. The closed wall mask is reduced to a one-pixel-wide centerline via the Zhang–Suen parallel thinning algorithm [19]. The resulting skeleton is the substrate for all subsequent geometry extraction.

Step 4: Vertex detection. In our wall graph, *vertices* represent either wall endings, or wall corners. Accordingly, vertices are computed using two different operators:

(1) *Topological analysis.* A 3×3 neighborhood kernel is convolved with the skeleton image. Pixels with ≥ 3 skeleton neighbors are classified as junctions; pixels with exactly one neighbor are classified as endpoints.
(2) *Geometric analysis.* The algorithm walks along skeleton chains and measures the local angle using a look-ahead/look-back window of $[5, 10]$ px. Pixels at which the turning

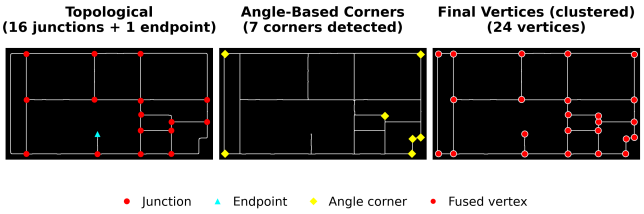


Figure 5: Topological (junctions, endpoints) and geometric (angle-based corners) vertex candidates fused into the final graph vertices.

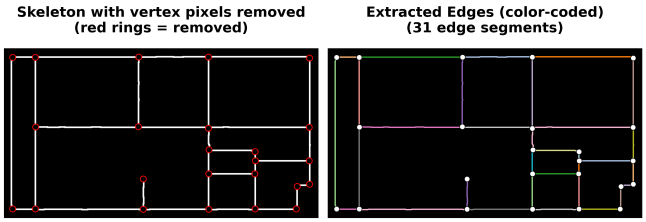


Figure 6: Each centerline segment between vertices is converted to an edge with ordered pixel geometry and arc-length parameterization.

angle exceeds 145° are classified as corners. Non-maximum suppression is used to eliminate redundant detections.

Topological and geometric vertex candidates are merged and any pair within a 15-pixel radius is fused into a single vertex (Fig. 5).

Step 5: Edge extraction. Wall graph *edges* represent pairs of connected vertices. They are obtained by simply removing points labeled as vertices from the skeleton images; the remaining connected components are labeled as edge segments. Each segment is traced as an ordered pixel chain from one endpoint vertex to the other, producing an `ordered_pixels` sequence with a (cumulative) `arc_length` value (Fig. 6).

At this stage, we have defined a (planar) wall graph, with vertices (wall corners or endings) connected by wall segments, and an *embedding* (a specific drawing) of this graph. It is also useful to identify the *dual* of the wall graph [5, Ch. 4], which is a graph whose vertices are the *bounded faces* of the wall graph embedding (in practice, these are rooms in the building). Two vertices of the dual graph (rooms) are connected if there is an edge of the primal graph (a wall) joining the two. Alternatively, by only defining edges between dual graph vertices when they communicate with each other through a door/opening, we obtain the *traversability graph* of the building.

In the following, we describe how to identify individual rooms and walls separating neighboring rooms, thereby defining the dual graph. Although, as mentioned above, rooms (i.e., areas delimited by doors and openings) can generally be identified as the faces of the wall graph, we found that some cases (in particular, nested rooms) complicate algorithms that simply reason on the planar graph structure to find faces. We leverage the information provided by VLM normalization by first clustering pixel colored in blue (representing "room interior" points), then finding individual faces

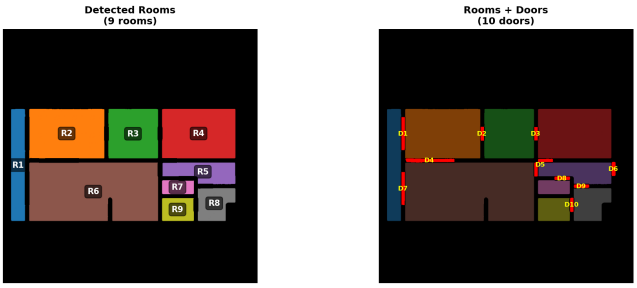


Figure 7: Each connected room region is labeled with a RoomID placed at its distance-transform centroid; door blobs are labeled separately on the right.

in the planar graph, and finally associating each such face with the correct blue pixel cluster.

Step 6: Room pixel clustering. The initial Semantic Normalization step imputes pixels belonging to room interiors by drawing them in blue color. We enumerate individual "blobs" of blue pixels, computed via 4–connected component analysis. For each room blob, we compute a centroid at the *deepest interior point* of the room: the argmax of the distance transform of the room mask. Each blob is given a unique RoomID starting from R1 to RN, where N is the number of rooms (Fig. 7). The centroid serves as the room's anchor point: it is used as the endpoint for the dual-graph and traversability-graph edges introduced above.

Step 7: Edge to room matching. Here we use the planar wall graph structure to determine and enumerate faces in the graph, then label each face against the semantic room blobs from Step 6, and propagate face labels to the edges that bound them. The procedure is two steps.

Step 7.1: Planar face enumeration via half-edge traversal. Formally, a face of a planar graph embedding is a maximal connected component of points in the plane that is disjoint from the graph. Faces can be computed following this simple algorithm. Each undirected wall edge produces two directed half-edges, one for each side of the wall. Tracing one face uses the half-edges on that face's side; the opposite half-edges remain unvisited and will trace the neighboring face on a later pass. To discover all faces, the algorithm iterates over the half-edges: each time it encounters an unvisited one it starts a new walk that, at every vertex, takes the sharpest clockwise turn available, which keeps the current face on a consistent side until the walk returns to where it started. This continues until every half-edge has been visited, at which point every face has been found (Fig. 8) [5, Ch. 4].

Step 7.2: Face labeling and edge inheritance. Each discovered face is a directed cycle of half-edges. To associate it with one blob of room interior points computed in Step 6 , the algorithm probes from the midpoint of each half-edge perpendicular to its walking direction: for an interior face (clockwise walk) the probe is fired inward, and for the exterior face (counterclockwise walk) it would be fired outward. Each probe is fired at a small set of short pixel distances and is labeled with the ID of the first room interior blob it

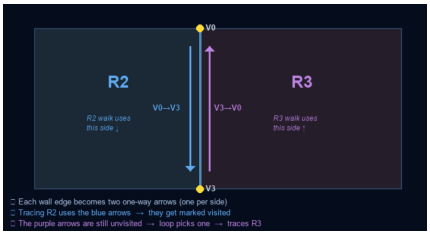


Figure 8: Each wall edge becomes two directed half-edges. Tracing face R_2 uses the blue half-edges; the purple half-edges on the opposite side are still unvisited and will trace face R_3 next.

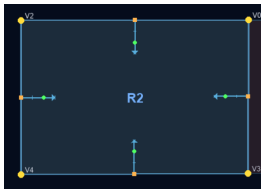


Figure 9: Each half-edge of a face fires a probe ray inward at short pixel distances; all probes hit the R_2 region from the semantic room labels of Step 6, so the face is labeled R_2.

hits. The majority label across all half-edges of the face determines the face's room ID. The exterior face (the set of planar points in the exterior of the planar graph embedding), which does not correspond to any room, receives no hits and is discarded.

Once each face is labeled, its boundary edges inherit that face's room ID. An edge shared by two faces (the wall between two adjacent rooms) receives both room IDs, so that geometric adjacency is recorded symmetrically on every shared wall (Fig. 9).

Step 8: Vertex inheritance. Vertices inherit room membership from their connected edges: if edge E belongs to room R, then the endpoints of E are added to the boundary vertex set of R.

Step 9: Door to edge mapping. Each labeled door blob is projected onto the corresponding skeleton edge and parameterized as a contiguous interval along the edge's arc length, `start_prop` to `end_prop`. This mapping comprises two steps.

PCA directed ray probing: For each door blob, the centroid and principal component axis are computed via PCA. Eight rays are fired in priority order: perpendicular to the door axis first (since the skeleton runs through the wall, which is perpendicular to the door opening), then along the door axis, then along the four diagonals. Each ray walks pixel by pixel, and the first ray to hit a skeleton edge identifies the host edge (Fig. 10).

Perpendicular probing for door extent: Every pixel of the host edge's `ordered_pixels` chain is walked in sequence; at each pixel a perpendicular ray is fired in both directions. If any ray hits the specific door blob, the pixel is marked as "in the door." Contiguous runs of in door pixels are converted to arc length proportions (`start_prop`, `end_prop`) $\in [0, 1]$, allowing multiple doors per edge to be represented (Fig. 11).

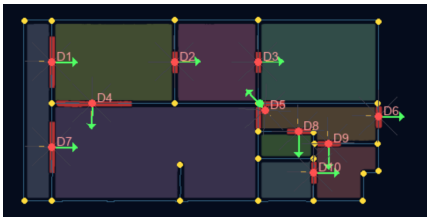


Figure 10: PCA-directed ray probing from door centroids to skeleton edges. Perpendicular-to-door rays are fired first and hit the host wall edge before any other ray direction.

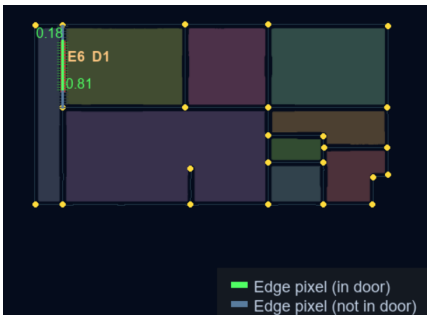


Figure 11: Perpendicular probing along the host edge's pixel chain determines door extent as arc-length proportions. Here door D_1 on edge E_6 spans the interval $0.18 \rightarrow 0.81$ of the edge's arc length.

3.3 Stage D: Structured Output

The final analysis view exposes summary cards (rooms, doors, vertices, edges, shared edges, edges-with-doors) and a diagnostic visualization. Programmatically, SIM 2.0 emits a JSON object encoding the full semantic graph. Rooms are keyed by integer ID and contain the centroid coordinates, connected doors, connected rooms, boundary vertices, and boundary edges. Edges contain vertex-index pairs, adjacent room IDs, pixel count, and door segments parameterized as arc-length proportions. For example:

```
{
  "rooms": {
    "1": { "id":1,
           "centroid":[234,156],
           "connected_doors":[3,7],
           "connected_rooms":[2,4],
           "boundary_vertices":[0,1,5,8],
           "boundary_edges":[1,3,7,12] }
  },
  "edges": [
    { "id":1, "vertex_indices":[0,1],
      "rooms":[1,2], "pixel_count":150,
      "door_segments":[
        { "door_id":3, "start_prop":0.3,
          "end_prop":0.7 } ] }
  ],
  "vertices": [[120,45],[340,45],
               [340,200],[120,200]]
}
```

4 Evaluation

4.1 Datasets and Protocol

We evaluate on 200 floor plans drawn from two established benchmark splits: the full 100-image test set of Raster-to-Vector [14] and a fixed 100-image subset sampled from the 500-image Raster-to-Graph test set [9]. Full-set evaluation on Raster-to-Graph was infeasible at N=24 due to API cost; both the Raster-to-Graph baseline and SIM 2.0 are evaluated on the identical fixed 100-image subset to ensure a fair comparison. Both Raster-to-Vector and Raster-to-Graph datasets are based on the LIFULL HOME'S dataset and can be acquired upon request from LIFULL Co., Ltd. via the NII Informatics Research Data Repository [13]. All Gemini inferences were performed via the Google AI Studio API using the `gemini-3-pro-image-preview` endpoint at sampling temperature 1.0 and top-p 0.95.

Two settings are reported for SIM 2.0: N=24 (full self-consistency) and N=1 (single Gemini sample, ablation). The metric definitions used throughout this section are formalized in Sec. 4.2 below.

4.2 Evaluation Metrics

We evaluate our floorplan vectorization across six categories: Vertices, Edges, Room Regions, Doors, Edge Connectivity, and Door connectivity. The first three capture geometric structure, the fourth captures functional elements, and the last two capture topological relations. Both the predicted and ground-truth RGB masks are run through the same vectorizer before scoring, so any vectorization artifact appears equally on both sides. All coordinates are normalized to the unit square $[0,1]^2$, distances are Euclidean, and a single tolerance $\tau = 0.01$ is used for every point and segment criterion. For each category we report precision, recall, and F_1.

- **Vertex.** For each ground-truth vertex, we find the closest unmatched predicted vertex and accept the match when their Euclidean distance is at most τ. Each predicted vertex can be matched at most once.
- **Edge.** A predicted edge matches a ground-truth edge when both of its endpoints/vertices lie within τ of the corresponding ground-truth endpoints under at least one of the two possible endpoint orderings. When more than one predicted edge qualifies for the same ground truth, we keep the one with the smallest sum of endpoint distances.
- **Door.** Doors are scored with the same matcher as edges, applied to door segments instead of wall segments.
- **Room Region.** Each room polygon is rasterized to a binary mask, and we compute IoU against the ground-truth room masks. A predicted room is accepted when its IoU with some unmatched ground-truth room exceeds 0.7. The accepted matches form the room correspondence M used by the two connectivity metrics below.
- **Edge Connectivity.** Two rooms are *wall-adjacent* when they share at least one edge. A ground-truth wall-adjacent pair is counted correct when both of its rooms appear in M and the corresponding pair of predicted rooms is also wall-adjacent.

Table 1: Comparison against Raster-to-Graph [9] on the 100-image subset of the Raster-to-Graph test set. SIM 2.0 (N=24) exceeds the baseline on every shared category. The N=1 ablation, which removes the self-consistency aggregation, drops F1 by 19–25 points, isolating the contribution of the voting stage.

	Raster-to-Graph			SIM 2.0 (N=24)			N=1
Category	Prec	Rec	F1	Prec	Rec	F1	F1
Room region	96.44	95.03	**95.73**	98.7	97.9	**98.3**	74.8
Vertices	98.53	97.80	**98.16**	98.7	99.2	**98.9**	80.2
Edges	96.70	95.84	**96.27**	98.0	98.1	**98.0**	74.9
Doors	–	–	–	96.2	91.6	93.9	71.4
Edge-conn.	–	–	–	97.5	96.5	97.0	71.7
Door-conn.	–	–	–	96.5	95.4	96.0	71.3

Table 2: Comparison against Raster-to-Vector [14] on the 100-image Raster-to-Vector test set. Edge connectivity and door-connectivity are not produced by Raster-to-Vector.

	Raster-to-Vector			SIM 2.0 (N=24)			N=1
Category	Prec	Rec	F1	Prec	Rec	F1	F1
Room region	85.2	89.9	87.5	98.5	98.6	**98.5**	73.9
Vertices	94.7	91.7	93.2	97.7	98.8	**98.3**	79.5
Edges	–	–	–	96.9	97.7	97.3	74.2
Doors	91.9	90.2	91.0	97.8	94.2	**95.9**	70.1
Edge-conn.	–	–	–	97.6	97.8	97.7	67.3
Door-conn.	–	–	–	97.6	97.7	97.6	63.7

- **Door Connectivity.** Two rooms are *door-adjacent* when they share a skeleton edge that carries at least one door segment. A ground-truth door-adjacent pair is counted correct when its image under M is itself door-adjacent.

4.3 Comparison with Raster-to-Graph

Table 1 reports results on the Raster-to-Graph test subset. SIM 2.0 with N=24 exceeds the Raster-to-Graph baseline on every shared category: +2.6 F1 on rooms, +0.7 F1 on vertices, and +1.7 F1 on edges. On the additional categories that the baseline does not produce (doors, edge-connectivity, door-connectivity) SIM 2.0 attains 93.9, 97.0, and 96.0 F1, respectively. The single-sample ablation (N=1) drops F1 by 19–25 points across the board, isolating the contribution of the self-consistency aggregation as the dominant source of reliability.

4.4 Comparison with Raster-to-Vector

Table 2 reports results on the Raster-to-Vector test set. SIM 2.0 attains 98.50 F1 on rooms (vs. 87.5 for the baseline, a +11.0 F1 improvement), 98.30 F1 on vertices (vs. 93.2, +5.1 F1), and 95.90 F1 on doors (vs. 91.0, +4.9 F1). The N=1 ablation again drops F1 by roughly 19–34 points across all categories, consistent with the Raster-to-Graph subset.

4.5 Comparison with Direct VLM Vectorization

A natural question is whether the deterministic vectorization pipeline of Sec. 3.2 is necessary at all: if Gemini is capable enough to produce a clean RGB semantic mask, can it also produce the final structured graph directly, in a single prompt? This subsection tests that hypothesis against a one-shot *Direct VLM Vectorization* (*DV* for short) baseline that asks the model to emit the full vector graph, for example in JSON format.

Protocol. Each of the 100 floor plans in the Raster-to-Vector test set is passed to `gemini-3.1-pro-preview` exactly once, with the prompt reported below. We found gemini-3.1-pro-preview instead to be more powerful than gemini-3-pro-image-preview for this particular task. The prompt we used is:

> *"Please extract the coordinate data for vertices, walls, doors, and room regions of this floorplan. Provide the output in a structured JSON format."*

Because the returned JSON's schema is non-deterministic (e.g. across runs we observed `walls` emitted with keys `start`/`end`, with keys `startVertexId`/`endVertexId`, or as bare index pairs, etc.), we issue a second call to the same model asking it to rewrite its first response under a strict fixed schema, reducing the problem to schema normalization. The second prompt is:

> *"You will receive a JSON description of a floorplan extraction (vertices, walls, doors, rooms / regions) produced in an arbitrary schema. Rewrite it as ONE STRICT JSON object with EXACTLY this shape and nothing else:*
>
> ```
> {
>   "vertices": [[x, y], ...],
>   "edges":    [[i, j], ...],
>   "rooms":    [{"name": "...", "vertices": [i,
> j, k, ...]}]
> }
> ..."
> ```

Because the second call sees only the first call's text, never the image, it can reformat but cannot invent geometry; we therefore still call this baseline one-shot "Direct VLM Vectorization".

We report two operating points for this DV baseline: a *threshold-parity* setting that uses the same thresholds as Table 2 (room IoU>0.7; vertex/wall endpoint τ=0.01), and a deliberately more *lenient* setting (room IoU>0.5; τ=0.05) that loosens the match criteria by a factor of 5× on τ. Results with the more lenient setting are reported only to highlight how much of the gap is attributable to threshold strictness vs. blatant structural error.

Table 3 reports results on the 100-image Raster-to-Vector test set across three categories: rooms, vertices, and edges. When the vanilla baseline is scored with the same match thresholds used for SIM 2.0 in Table 2 (room IoU>0.7 and endpoint τ=0.01), it reaches only 31.3, 22.9, and 10.3 F1 on rooms, vertices, and edges respectively, trailing SIM 2.0 by 67.2, 75.4, and 87.0 F1 points. Relaxing the thresholds 5× on τ (to 0.05) and lowering the room IoU requirement from 0.7 to 0.5 recovers an additional 20.1, 45.3, and 40.3 F1 points on rooms, vertices, and edges. These are meaningful, but the relaxed scores remain 30.1 to 47.1 F1 points short of SIM 2.0 across all three categories. The lenient setting is therefore an upper bound on what threshold relaxation alone can buy; the residual gap visible in Table 3 is arguably structural error. The final DV Gemini JSON output tends to drop rooms and emit walls that are geometrically inconsistent with the input, while the SIM 2.0 consensus mask generally preserves the full structure.

Table 3: Comparison against Direct VLM Vectorization (DV) using `gemini-3.1-pro-preview` on the 100-image Raster-to-Vector test set. The middle column block reports the same baseline with deliberately relaxed match thresholds (room IoU>0.5, endpoint τ=0.05). The SIM 2.0 column is reproduced from Table 2.

	DV (IoU>0.7, τ=0.01)			DV (IoU>0.5, τ=0.05)			SIM 2.0 (N=24) (IoU>0.7, τ=0.01)		
Category	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
Room region	35.6	27.9	31.3	58.5	45.8	51.4	98.50	98.60	**98.50**
Vertices	21.8	24.1	22.9	64.9	71.9	68.2	97.70	98.80	**98.30**
Edges	10.4	10.3	10.3	50.9	50.4	50.6	96.90	97.70	**97.30**

Table 4: Characterizing the N-sweep with respect to F1 on the 100-image Raster-to-Vector test set. Every category improves monotonically with N; the largest single jump is from N=1 to N=3, and the curve flattens past N=12.

Category	N=1	N=3	N=6	N=12	N=24
Room region	73.9	89.3	93.4	97.1	**98.5**
Vertices	79.5	91.0	94.1	97.0	**98.3**
Edges	74.2	87.2	90.6	95.3	**97.3**
Doors	70.1	80.8	86.5	92.8	**95.9**
Edge-conn.	67.3	84.9	89.4	95.1	**97.7**
Door-conn.	63.7	83.5	89.2	94.4	**97.6**

4.6 N-Sweep: Effect of Sample Count on Reliability

Tables 1 and 2 report the two endpoints of the self-consistency aggregation, N=1 and N=24. Table 4 fills in the curve at $N \in \{1, 3, 6, 12, 24\}$ on the 100-image Raster-to-Vector test set, so that the marginal value of each additional Gemini sample can be read off directly.

Three trends are visible. *First*, every category improves monotonically with N, confirming that the pixel-wise vote is doing useful work at every sample count rather than merely averaging equivalent draws. *Second*, the largest single gain is from N=1 to N=3 (between +10.7 and +19.8 F1 across categories), so even a modest amount of aggregation removes the bulk of per-sample hallucinations; the curve then flattens, with the N=12→24 step contributing only 1.3–3.2 F1 points. *Third*, the topological metrics (edge- and door-connectivity) gain the most from the sweep, roughly 30–34 F1 points from N=1 to N=24, because a single mislabeled wall or hallucinated door can break a connectivity edge and the vote suppresses exactly those low-frequency, sample-specific errors. The choice of N=24 in the main comparisons is therefore a saturation point rather than an arbitrary budget: practitioners trading cost for reliability can drop to N=12 for $\approx$ 1–3 F1 of degradation, or to N=6 for a larger but still usable $\approx$ 4–9 F1 loss.

4.7 Failure Modes of Non-Foundational Models

The aggregate metrics of Sec. 4 obscure a structural property of our comparison: both Raster-to-Vector and Raster-to-Graph are end-to-end supervised systems whose accuracy is bounded by the distribution and resolution of their narrow-scope training data sets. SIM 2.0 relies on a foundational VLM that was trained over a massive amount of diverse and unstructured data. In addition, these prior models by their own nature required a fixed input image size. Raster-to-Vector resamples any given input to 256×256 [14] pixels, and Raster-to-Graph requires a centered 512×512 crop [9]. These sizes are structural: the convolutions, the transformer's positional embeddings, and the autoregressive decoder's coordinate frame are all built around them. Squashing some floorplans to 512×512 (or 256×256) collapses walls to one or two pixels and erases narrow walls before the network sees them making the models useless for bigger floor plans. In contrast, SIM 2.0 accepts input images of any size. This is because the Gemini VLM at the core of our pipeline leverages a dynamic tiling strategy for tokenization [8]. Rather than globally downsampling high-resolution images, the model dynamically scales and divides the input into a grid of 768×768 pixel tiles. Each tile is independently tokenized, ensuring that fine-grained architectural details are preserved at high fidelity without being degraded by forced compression. The subsequent geometric vectorization step is designed to support any image size. In particular, pixel-domain thresholds of Sec. 3.2 (corner non-maximum-suppression radius, minimum chain length, room-area floor, dilation iterations) are calibrated at a fixed reference dimension and scale linearly with the longer side of the input.

The gap between SIM 2.0 and prior models is highlighted in Fig. 12, which shows three seemingly simple floor plans, on all of which both Raster-to-Vector and Raster-to-Graph cannot produce a meaningful output (they crash or produce an output that is blank or contains one or two lines). SIM 2.0 succeeds in all three cases, though possibly with a few errors. The first floor plan in Fig. 12 was taken from AdobeStock. It is geometrically straightforward, but the wall color and the background color differ from the conventions on which Raster-to-Vector and Raster-to-Graph were trained. SIM 2.0 faithfully rendered this plan in vectorized form, though it missed one small isolated vertical wall visible at the center of the image (flanked by two openings). The second floor plan in the figure was sketched with an Apple Pencil on an iPad. The third example is a hand drawn map (from the web site evstudio.com). SIM 2.0 was able to vectorize it, with only a few mistakes (two "ghost" doors added). These floor plan examples have different "style" than those used to train Raster-to-Vector or Raster-to-Graph, highlighting the sensitivity of these systems to the specific graphic conventions used to draw the maps.

4.8 Discussion

Our experiments showed that, when tested on the same data sets used in the evaluations of baseline algorithms (Raster-To-Graph and Raster-To-Vector), SIM 2.0 produces more accurate results on all considered metrics. More importantly, SIM 2.0 works well on different map rendering styles (where the baseline methods fail), and with different input image sizes (not supported by the baseline algorithms). Remarkably, these results were obtained *without* the need to retrain the system. The semantic-interpretation layer inherits the open vocabulary generality of a frontier VLM, and the geometry layer, operating on a constrained four-class mask, is by construction style- and resolution- invariant.

Our 2-step approach to vectorization may at first sight seem redundant: after all, VLMs have the capability to *directly* generate a vectorized output, when prompted appropriately. Yet, our experiments have shown that direct vectorization (DV) produced much poorer results than our 2-step method. It is possible that, by applying self-consistency [18], DV may improve its accuracy, as was the case for our system. One issue that would need to be resolved, however, is how to integrate multiple versions of a structured (e.g. JSON) vectorized output to obtain something resembling majority vote. For example, two generated samples could differ in the number of detected structural elements, and it is not clear how these could be combined together. In SIM 2.0, self-consistency was applied to a style transfer task, generating N images, all with the same size, each pixel of which could only represent one of 4 classes. This suggested our straightforward (and successful) approach of per-pixel majority vote computation.

4.9 Limitations

The dominant limitation of the SIM 2.0 pipeline is API cost. To obtain accurate results, the first semantic-interpretation layer (style transfer) needs to be run N times in a self-consistency strategy. We thus trade cost for reliability, but for very large datasets, the cost may be non-trivial. A second limitation is our choice of a four-class semantic vocabulary; for example, staircases and other structured elements are mapped to walls or background. Additionally, room type identification (e.g., bathroom vs. bedroom) is also not currently supported. Extending the structural vocabulary and enriching the semantics of structural elements require a richer style reference and is left for future work.

5 Conclusion

We have presented **SIM 2.0**, a one-shot, training-free pipeline for the conversion of architectural floor plan images into structured vector graphs. The key architectural decision, normalizing diverse floor plan formats into an RGB semantic mask before deterministic vectorization, enables stable extraction behavior across heterogeneous input sources. Stochastic VLM artifacts that would otherwise corrupt downstream geometry are suppressed by a pixel-wise self-consistency vote across N=24 samples; an ablation shows that this aggregation is the dominant source of reliability in the system. The deterministic vectorization stage uniquely produces both halves of a representation that prior work splits apart: the geometric primitives (wall skeletons, vertices, and room polygons) that Tesseract [3] leaves out, together with a door-level traversability graph that prior geometric vectorization systems do not output.

On the standard Raster-to-Vector test set and a 100-image subset of the Raster-to-Graph test set, SIM 2.0 exceeds both supervised baselines on every shared metric without any labeled training data.

Future work includes extending the semantic mask vocabulary beyond four classes (stairs, icons, room types), and supporting multi-story floor plans with inter-floor connectivity.

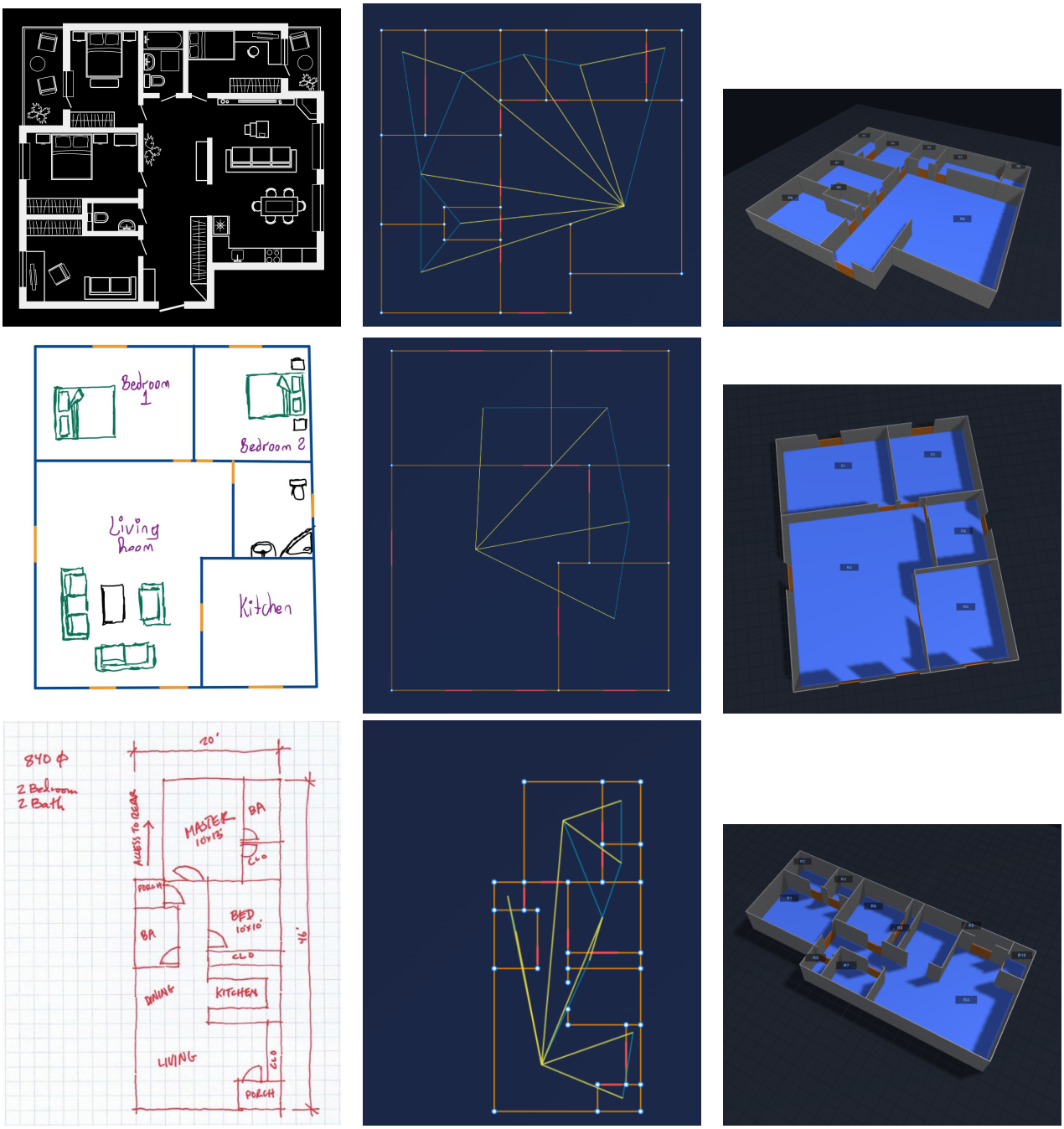


Figure 12: Three examples where Raster-to-Vector and Raster-to-Graph cannot produce a meaningful output. Left: the image floor plan input (respectively, an AdobeStock sample, a map drawn on an iPad using an Apple Pencil, and a hand-drawn plan (from evstudio.com). Middle: the vectorized output from SIM 2.0, shown with its traversability graph. Right: 3D simulation rendered from the SIM 2.0 vectorized output.

Acknowledgments

This research was funded by the National Eye Institute, National Institutes of Health, grant number R01EY035433. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

References

[1] Sheraz Ahmed, Marcus Liwicki, Markus Weber, and Andreas Dengel. 2011. Improved automatic analysis of architectural floor plans. In *2011 International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 864–869.

[2] Sheraz Ahmed, Marcus Liwicki, Markus Weber, and Andreas Dengel. 2012. Automatic room detection and room labeling from architectural floor plans. In *2012 10th IAPR international workshop on document analysis systems*. IEEE, 339–343.

[3] Yaqoob Ansari, Ammar Karkour, Eduardo Feo Flushing, and Khaled A. Harras. 2025. Tesseract: Unfolding Navigable Graph Representations from Low-Semantic Floor Plans. In *Proceedings of the 33rd ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL '25)*. ACM, 569–579.

[4] Lluis-Pere De Las Heras, David Fernández, Ernest Valveny, Josep Lladós, and Gemma Sánchez. 2013. Unsupervised wall detector in architectural floor plans. In *2013 12th International Conference on Document Analysis and Recognition*. IEEE, 1245–1249.

[5] Reinhard Diestel. 2017. *Graph Theory* (5 ed.). Springer.

[6] Samuel Dodge, Jiu Xu, and Björn Stenger. 2017. Parsing floor plan images. In *2017 Fifteenth IAPR international conference on machine vision applications (MVA)*. IEEE, 358–361.

[7] Lucile Gimenez, Sylvain Robert, Frédéric Suard, and Khaldoun Zreik. 2016. Automatic reconstruction of 3D building models from scanned 2D floor plans. *Automation in Construction* 63 (2016), 48–56.

[8] Google. 2026. Understand and count tokens: Multimodal tokens. https://ai.google.dev/gemini-api/docs/tokens?lang=python#multimodal-tokens. Gemini API documentation. Accessed 2026-06-02.

[9] Sizhe Hu, Wenming Wu, Ruolin Su, Wanni Hou, Liping Zheng, and Benzhu Xu. 2024. Raster-to-Graph: Floorplan Recognition via Autoregressive Graph Prediction with an Attention Transformer. *Computer Graphics Forum* 43, 2 (2024), e15007.

[10] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1125–1134.

[11] Ahti Kalervo, Juha Ylioinas, Markus Häikiö, Antti Karhu, and Juho Kannala. 2019. CubiCasa5K: A dataset and an improved multi-task model for floorplan image analysis. In *Scandinavian Conference on Image Analysis*. Springer, 28–40.

[12] Seongyong Kim, Seula Park, Hyunjung Kim, and Kiyun Yu. 2021. Deep floor plan analysis for complicated drawings based on style transfer. *Journal of Computing in Civil Engineering* 35, 2 (2021), 04020066.

[13] LIFULL Co., Ltd. 2015. LIFULL HOME'S Dataset. Informatics Research Data Repository, National Institute of Informatics. doi:10.32130/idr.6.0 Dataset.

[14] Chen Liu, Jiajun Wu, Pushmeet Kohli, and Yasutaka Furukawa. 2017. Raster-to-vector: Revisiting floorplan transformation. In *Proceedings of the IEEE international conference on computer vision*. 2195–2203.

[15] Sébastien Macé, Hervé Locteau, Ernest Valveny, and Salvatore Tabbone. 2010. A system to detect rooms in architectural floor plan images. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*. 167–174.

[16] Viet Trinh and Roberto Manduchi. 2019. Semantic interior mapology: A toolbox for indoor scene description from architectural floor plans. In *24th International Conference on 3D Web Technology*. 1–2. doi:10.1145/3329714.3338141

[17] Viet Trinh, Roberto Manduchi, and Nicholas A Giudice. 2023. Experimental evaluation of multi-scale tactile maps created with SIM, a web app for indoor map authoring. *ACM Transactions on Accessible Computing* 16, 2 (2023), 1–26.

[18] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*. https://openreview.net/forum?id=1PL1NIMMrw

[19] T. Y. Zhang and C. Y. Suen. 1984. A fast parallel algorithm for thinning digital patterns. *Commun. ACM* 27, 3 (1984), 236–239.