# HOUSEDIFFUSION++: CONSTRAINT-CONSISTENT VECTOR FLOORPLAN GENERATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

In real architectural design scenarios, floorplans must satisfy a variety of requirements imposed by regulations, design briefs and project-specific constraints. Often combining geometric, functional, and semantic requirements, these conditions make practical floorplan generation a highly constrained design problem rather than an unconstrained layout synthesis problem. Current floorplan generation methods usually address only a small subset of these conditions. In this work, we introduce HouseDiffusion++, a constraint-consistent diffusion framework for controllable vector floorplan generation. Building on HouseDiffusion, our approach extends vector denoising with several conditioning mechanisms to enable joint control over room adjacency graphs, outer boundaries, room sizes, windows and entrance doors. We also address a significant mismatch between training and inference in the original formulation, whereby the number of room corners is known during training but unavailable at test time. To overcome this issue, we introduce a polygon augmentation strategy that enhances robustness to redundant points. Experiments on the RPLAN dataset demonstrate that HouseDiffusion++ outperforms baselines in terms of quantitative metrics and visual quality, while offering greater controllability under different constraints.

## 1 INTRODUCTION

Automatic floorplan generation aims to produce layouts that combine plausible spatial organization with user-specified design requirements (Weber et al., 2022; Liu et al., 2025; Meselhy & Almalkawi, 2025). Public datasets and studies of architectural-plan data further emphasize that practical floorplan generation must account for geometric validity, representation quality, and heterogeneous design constraints rather than visual plausibility alone (Kalervo et al., 2019; van Engelenburg et al., 2024; Carrera et al., 2024; Mostafavi et al., 2025). In residential design, room connectivity, the apartment boundary, target room areas, and existing openings must be considered jointly. These conditions interact: enlarging one room within a fixed boundary changes the space available to its neighbors and may affect their access to windows. A useful generator must therefore coordinate these requirements while producing editable geometry for subsequent design stages.

Prior work has made substantial progress in conditional floorplan generation. Existing models address individual requirements, such as room connectivity or the outer boundary (Nauata et al., 2021; Wu et al., 2019), while other frameworks combine several constraints to provide broader control over the generated layout (Hu et al., 2020; Sun et al., 2022; Zhang et al., 2024). For practical architectural use, supporting the required conditions must be accompanied by high geometric quality: artifacts in generated layouts can require substantial correction before further editing in architectural software. Jointly satisfying these conditions while producing realistic and geometrically consistent vector layouts remains an open challenge.

We introduce HouseDiffusion++, a unified diffusion framework for vector floorplan generation. Building on HouseDiffusion (Shabani et al., 2023), our model combines spatial conditioning on the apartment boundary and prescribed openings with conditioning on target room areas, alongside the room adjacency graph. These inputs jointly guide the denoising of vector coordinates. A fundamental limitation of the underlying representation is that the number of points assigned to each room must be chosen before generation. At inference, the true corner count is unknown, and surplus points can introduce spurious corners and geometric artifacts. We address this sensitivity through a



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

polygon augmentation strategy that inserts redundant points along room edges during training without changing their geometry. This encourages the model to accommodate excess points on existing walls, allowing a simple fixed-budget policy to approach the generation quality obtained with ground-truth corner counts in our experiments.

Our main contributions are:

- **Jointly conditioned vector diffusion.** To our knowledge, we propose the first unified diffusion framework for vector floorplan generation that jointly conditions on room adjacency, the apartment boundary, target room areas, and prescribed window and entrance-door geometry.
- **Robustness to point-budget uncertainty.** We introduce a polygon augmentation strategy that reduces sensitivity to the number of points allocated per room and substantially narrows the quality gap between fixed-budget inference and inference with ground-truth corner counts.
- **Empirical evaluation.** Comparisons with five publicly available baselines on RPLAN demonstrate improvements in generation quality and constraint adherence in the evaluated settings. Controlled ablations assess the conditioning mechanisms and isolate the effect of augmentation on robustness to the inference budget.

## 2 Related Works

**Image-based floorplan generation.** Image-based methods synthesize floorplans in a raster representation, where pixels encode spatial regions and architectural elements such as rooms, walls, and openings. Early approaches generate room arrangements within a given boundary (Wu et al., 2019) or synthesize and refine room masks conditioned on an adjacency graph (Nauata et al., 2021). GAN-based raster formulations were subsequently extended through ActFloor-GAN, FLNet, and FloorGAN, which incorporate additional structural or graph-based conditioning while retaining image-like output representations (Wang et al., 2023; Upadhyay et al., 2022; 2023). More recent methods broaden the range of user inputs: ChatHouseDiffusion (Qin et al., 2024) combines an outer contour with textual specifications of room connectivity, size categories, and locations, while MaskPLAN (Zhang et al., 2024) completes layouts from partially specified room types, positions, adjacency, areas, and regions within a given boundary. Hybrid approaches also support multiple conditions. Rahbar et al. (2022) combine agent-based modeling with a conditional GAN to generate layouts from a footprint, a connectivity graph, desired room areas, and prescribed windows and an entrance. These requirements guide a bubble diagram that is converted into a heat map, from which the GAN predicts a raster layout. These methods demonstrate that a broad set of design conditions can be integrated within a generation pipeline. Recent raster-oriented systems also scale to larger layouts or combine generation and editing in multimodal formulations, including CG-Floor (Lian et al., 2026) and HouseMind (Qin et al., 2026). Practical architectural design workflows typically require editable vector geometry. Raster predictions can be converted into this format, but vectorization may introduce additional geometric artifacts that require further correction.

**Vector floorplan generation.** Structured approaches represent layouts through room boxes, wall graphs, or polygon coordinates. FloorplanGAN (Luo & Huang, 2022) is an early learned vector-oriented formulation, while Liu et al. (2022) generate graph-constrained vectorized floorplans through draft generation followed by panoptic refinement. Graph2Plan (Hu et al., 2020) and the factor-graph formulation of Dupty et al. (2024) combine room relationships and geometric attributes with a prescribed boundary, predicting room boxes followed by layout refinement. WallPlan (Sun et al., 2022) generates wall graphs and supports boundary, window, load-bearing wall, and bubble-diagram constraints, including combinations of these inputs. Its bubble diagrams encode room adjacency, location, and relative size, making it a close predecessor in terms of supported design requirements. Diffusion-based approaches provide another route to vector generation. HouseDiffusion (Shabani et al., 2023) denoises room and door coordinates conditioned on an adjacency graph, while Cons2Plan (Hong et al., 2024) and the boundary-conditioned extension of Stoppani et al. (2026) support joint graph and boundary conditioning. GSDiff (Hu et al., 2025) generates wall junctions and predicts their connections, with separate evaluations under boundary and topology conditions. DiffPlanner (Wang & Pajarola, 2025) decomposes generation into room attributes, adjacency, and partitioning, allowing users to specify the boundary and partial room information, including sizes and

locations. TLC-Plan (Xiong et al., 2026) and CE2EPlan (Wang & Pajarola, 2026) further develop structured/vector layout generation under explicit constraints. Beyond diffusion, FMLM (Shiohara & Yamasaki, 2026) autoregressively generates vector layouts in a markup representation with joint boundary and graph conditioning. Language-based structured generation has also been explored in FloorPlan-LLaMA (Yin et al., 2025), Text-to-Layout (Duggempudi et al., 2025), and House-Tune (Zong et al., 2026), which encode layouts or editing operations as token sequences or structured textual representations. Despite this progress, jointly accommodating diverse design conditions while maintaining geometric quality remains challenging. Supporting the required inputs does not by itself ensure that the generated layouts satisfy them accurately or avoid geometric artifacts. Motivated by this challenge, we introduce a unified vector diffusion framework that jointly incorporates room adjacency, the apartment boundary, target room areas, and prescribed openings, and demonstrate improved generation quality and constraint adherence in our comparative evaluation.

# 3 METHOD

## 3.1 PRELIMINARIES: HOUSEDIFFUSION

Our method builds on HouseDiffusion (Shabani et al., 2023), a diffusion-based framework for vector floorplan generation. We therefore briefly review its representation and denoising architecture. Following HouseDiffusion, a floorplan is represented as a set of $N$ polygons

$$P = \{P_i\}_{i=1}^{N}, \qquad P_i = \{C_{i,j}\}_{j=1}^{N_i}, \quad C_{i,j} \in \mathbb{R}^2,$$

where each polygon corresponds to a room, door, or window, and $N_i$ is the number of its corners. During training, the ground-truth corner coordinates are corrupted with Gaussian noise; at inference, the model starts from random noise and iteratively denoises it into a vector floorplan.

HouseDiffusion treats each corner as a token in a Transformer denoising network, embedding its coordinates together with the corner index, polygon type, polygon index, and diffusion timestep. The network combines three attention mechanisms: component-wise self-attention (CSA) within each polygon, global self-attention (GSA) across the floorplan, and relational cross-attention (RCA) guided by the adjacency graph. At each denoising step, the model predicts the noise in the corner coordinates. An auxiliary 8-bit binary coordinate prediction branch further promotes geometric consistency, including shared corners and aligned walls.

We retain this polygon representation and denoising backbone while introducing spatial and room-area conditioning, together with a training augmentation that improves robustness to the number of points allocated at inference.

## 3.2 SPATIAL CONDITIONING

We represent the apartment *boundary*, prescribed *windows*, and *front door* as spatial polygons that condition the denoiser through a single unified pathway, illustrated in Fig. 1. This pathway incorporates both global boundary information and localized opening constraints.

The prescribed apartment boundary is expressed as a polygon $B = \{B_k\}_{k=1}^{N_B}$, where $B_k \in \mathbb{R}^2$ are its corners and $N_B$ is their number. We model each window $W_j$ as a polygon with 4 corners $w_{j,k} \in \mathbb{R}^2$, two of which lie on the exterior boundary edge it is attached to, and two of which extend inward perpendicular to that edge. The front door $F$ is likewise represented as a polygon with 4 corners attached to the apartment boundary. During dataset preparation, the boundary and front door are obtained from the annotated floorplans. Since the dataset lacks window annotations, windows are placed procedurally using Algorithm 1 (Appendix A). At inference, the spatial polygons are supplied as design conditions.

We unify the boundary, windows, and front door under a single set of spatial polygons:

$$C^{sp} \;=\; \big\{B\big\} \;\cup\; \big\{W_j\big\}_{j=1}^{N_W} \;\cup\; \big\{F\big\},$$

where $N_W$ is the number of windows. Following HouseDiffusion, for every corner $C_{i,j}$ of a polygon $P_i \in C^{sp}$, we uniformly sample $L = 8$ points along the edge toward the next corner. We concatenate the corner coordinates with those of the sampled points to form a single spatial token. The resulting

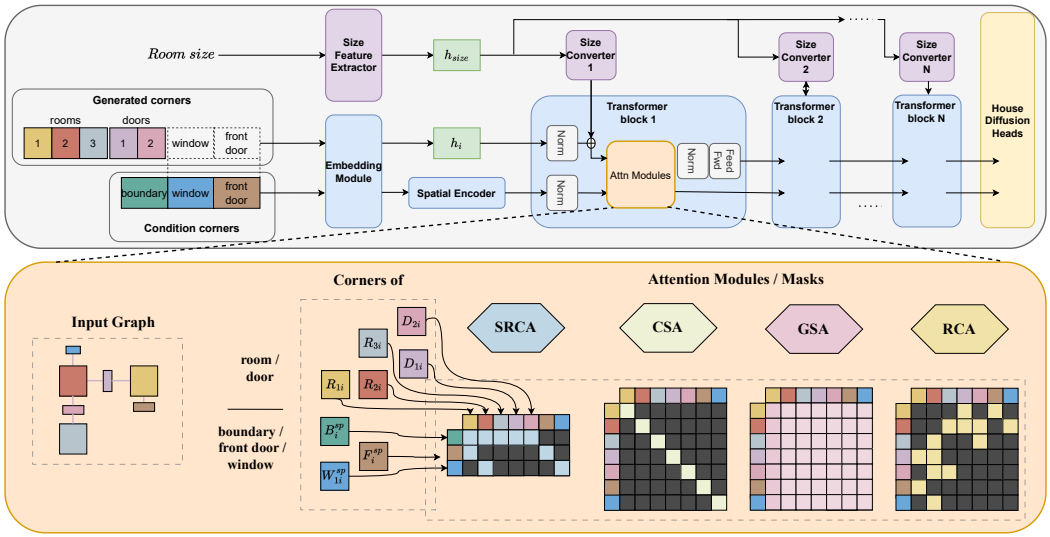


Figure 1: Overview of HouseDiffusion++. Generated room, door, and window corners are denoised by a Transformer, while spatial condition corners—the apartment boundary, windows, and front door—are encoded separately and injected through Spatial Room Cross-Attention (SRCA). Room-size constraints are provided as relative area ratios and incorporated in every Transformer block via Size Converters. The bottom panel shows the attention masks used by SRCA, CSA, GSA, and RCA to combine spatial constraints, polygon structure, global context, and graph relations.

representation is $E^{sp} \in \mathbb{R}^{N_C \times (L+1)\cdot 2}$, where $N_C$ is the total number of corners across the spatial polygons.

**Spatial Encoder.** Each spatial token $e_k^{sp} \in E^{sp}$ is linearly projected into the model dimension $d$. We then add a conditioning vector $h_k^{\text{cond}}$, computed as the sum of learned embeddings for the polygon type (boundary, window, or front door), polygon index, and corner index.

The resulting sequence $H^{sp} = [h_1^{sp}, \ldots, h_{N_C}^{sp}] \in \mathbb{R}^{N_C \times d}$ contains boundary, window, and front-door tokens, which lets us toggle window or front-door conditioning on or off by simply including or excluding the corresponding entries.

The spatial sequence $H^{sp}$ is then processed by a separate Spatial Encoder. It consists of several Transformer blocks with the same architecture as the main denoising model, but uses only global self-attention (GSA) over the spatial tokens. The resulting features $\hat{H}^{sp}$ encode the global geometry of the apartment shell together with the provided architectural openings, and are subsequently used by the main denoising network as a condition.

**Spatial Room Cross-Attention.** To inject the spatial condition into the denoising network, we introduce an additional cross-attention layer, Spatial Room Cross-Attention (SRCA), used in parallel with CSA, GSA, and RCA. SRCA attends from the main floorplan tokens to the spatial tokens $\hat{H}^{sp}$ produced by the Spatial Encoder. The denoising sequence contains tokens for rooms, doors, and windows, while the separate spatial tokens encode the prescribed boundary and opening geometry. When window or front-door geometry is not provided as a spatial condition, the corresponding elements are generated using the remaining available conditions. We use a spatial attention mask to control which condition tokens are visible to each part of the generated layout: boundary tokens are visible to all floorplan tokens, while window and front-door condition tokens are visible only to their corresponding generated tokens and to room tokens connected to them in the input graph. This guides the reverse diffusion process toward layouts that respect the prescribed apartment boundary while placing windows and the entrance door consistently with the provided condition.

### 3.3 Size Conditioning

HouseDiffusion represents floorplans in normalized coordinates, so absolute room areas depend on the physical scale assigned to the layout. We therefore express each target room area as a fraction of

the total apartment area. This provides a scale-independent condition that describes how the available space should be distributed among rooms. When the apartment area in physical units is known, target areas specified in square meters can be converted directly into these ratios. During training, the target area ratio for room $i$ is computed from the annotated floorplan as

$$s_i = \frac{\mathrm{Area}(P_i)}{\mathrm{Area}(B)}, \qquad s_i \in [0, 1],$$

where $P_i$ is the room polygon and $B$ is the apartment boundary. At inference, these ratios are supplied as user-specified conditions.

Each corner token $C_{i,j}$ receives the target area ratio $s_i$ of its corresponding room. A Size Feature Extractor maps these scalar values to size embeddings $h_{\mathrm{size}}$.

Each Transformer block has a separate Size Converter that transforms these shared embeddings into block-specific features, which are added to the corresponding token representations before attention. This incorporates room-area information throughout the denoising network. The architectures of the Size Feature Extractor and Size Converter are detailed in Appendix E and Fig. 6.

To strengthen adherence to the target areas, we use classifier-free guidance with respect to the size condition. During training, the size condition is dropped with probability $p$, in which case no size embedding is added. At inference, we evaluate the denoiser with and without size conditioning and combine its predictions as

$$\hat{x} = \hat{x}_{\mathrm{uncond}} + w\left(\hat{x}_{\mathrm{cond}} - \hat{x}_{\mathrm{uncond}}\right),$$

where $\hat{x}_{\mathrm{cond}}$ and $\hat{x}_{\mathrm{uncond}}$ denote the predictions with and without the size condition, respectively, and $w$ is the guidance scale. The adjacency graph and any provided spatial conditions are retained in both passes. We analyze the effect of the guidance scale in Sec. 4.3.

### 3.4 Data augmentation

HouseDiffusion requires the number of corners per room $N_i$ to be specified before generation. During training, this number is available from the ground-truth polygons, whereas at inference it must be sampled from dataset statistics or chosen manually. Allocating more points than expected provides capacity to represent rooms of different geometric complexity, but surplus points can introduce spurious corners and distorted boundaries. Our goal is to reduce sensitivity to this choice by allowing additional points to lie along existing walls without creating new geometric corners.

To this end, we augment room polygons with redundant points during training. For each edge $(C_{i,j}, C_{i,j+1})$ of a polygon $P_i$, we insert $k$ additional points with probability $p_k$, sampling their locations uniformly along the segment. The augmented polygon retains the original geometry while using a different number of points to represent it. Exposing the model to such alternative representations encourages it to accommodate redundant points along walls and reduces the tendency to interpret every point as a distinct geometric corner. This augmentation is applied only to room polygons; doors and windows remain unchanged.

At inference, this enables a simple fixed-budget policy in which each room is assigned a predetermined number of points without access to its ground-truth corner count. The budget is still specified in advance, but the augmentation reduces the need to match it precisely to the geometric complexity of each room. In Sec. 4.4, we evaluate sensitivity to the allocated budget and measure the quality gap relative to an oracle setting that uses ground-truth corner counts.

### 3.5 Geometric post-processing.

Generated vector layouts may contain geometric artifacts, such as self-intersections, overlapping rooms, or uncovered regions within the prescribed boundary, that complicate their use in architectural software. We therefore provide an optional deterministic post-processing procedure to repair invalid room polygons, resolve overlaps, and fill uncovered regions within the apartment boundary. All quantitative results and qualitative examples in the main paper are obtained without post-processing. The procedure and results illustrating its effect are presented in Appendix B.

Table 1: Quantitative comparison with publicly available baselines, without window or entrance-door conditioning. Conditions: *G* – room-adjacency graph; *B* – apartment boundary; *S* – target room areas. Best and second-best values are shown in bold and underlined, respectively.

| Method | Condition | FID ↓ | FIDgs ↓ | KID ↓ | KIDgs ↓ | BoundIoU ↑ | GED ↓ | SizeError ↓ |
|---|---|---|---|---|---|---|---|---|
| House-GAN++ | G | 25.33 | 34.93 | 23.08 | 38.84 | 0.583 | 2.38 | 0.523 |
| HouseDiffusion | G | 8.42 | 10.61 | 1.92 | 5.75 | 0.578 | 0.76 | 0.246 |
| GSDiff | G | 9.42 | 7.04 | 8.42 | 1.06 | 0.652 | – | 0.826 |
| GSDiff | B | 13.41 | 7.55 | 15.28 | 2.59 | 0.965 | – | – |
| WallPlan | B | 20.85 | 8.35 | 17.26 | 3.85 | **0.975** | 4.08 | – |
| Graph2Plan | B + G | 13.97 | 8.01 | 7.38 | 3.12 | 0.953 | 2.05 | 0.178 |
| Ours w/o size | B + G | <u>6.43</u> | <u>4.21</u> | **0.89** | <u>0.63</u> | 0.966 | <u>0.33</u> | <u>0.163</u> |
| Ours | B + G + S | **6.05** | **3.98** | <u>0.96</u> | **0.52** | <u>0.974</u> | **0.28** | **0.028** |

# 4 EXPERIMENTS

## 4.1 EXPERIMENTAL SETUP

We conduct experiments on the RPLAN dataset (Wu et al., 2019), which contains real residential floorplans represented with room polygons and semantic room labels. We use 62,000 floorplans for training, 900 floorplans for validation, and 600 floorplans for testing. All reported quantitative and qualitative results are computed on the test split. Unless otherwise stated, our model is conditioned on the room adjacency graph, apartment boundary, and target room areas.

All variants of our model are trained on two NVIDIA H100 GPUs. We use the Adam optimizer with a total batch size of 512. Each model is first trained for 200k iterations with a learning rate of $10^{-4}$, followed by an additional 100k iterations with a reduced learning rate of $10^{-5}$. For classifier-free guidance, the size condition is dropped during training with probability $p = 0.1$. For data augmentation, we use $p_0 = 0.9$, $p_1 = 0.04$, $p_2 = 0.03$, $p_3 = 0.02$, and $p_4 = 0.01$.

We compare our method with Graph2Plan (Hu et al., 2020), House-GAN++ (Nauata et al., 2021), WallPlan (Sun et al., 2022), HouseDiffusion (Shabani et al., 2023), and GSDiff (Hu et al., 2025). We restrict our comparisons to methods with publicly available implementations. Related methods for which code was unavailable at the time of evaluation are therefore not included. For WallPlan, we evaluate the boundary-conditioned variant available through the released inference pipeline; we could not locate public inference code for the additional conditioning modes described in the paper.

## 4.2 COMPARISON WITH EXISTING METHODS

**Evaluation metrics.** We evaluate generated floorplans using both distribution-level and condition-compliance metrics. To measure visual similarity to real layouts, we report Fréchet Inception Distance (FID) and Kernel Inception Distance (KID). We use Graph Edit Distance (GED) to evaluate consistency with the input room-adjacency graph, and Boundary IoU (BoundIoU) to measure agreement between the generated layout envelope and the given apartment boundary. To evaluate room-size conditioning, we introduce SizeError:

$$\text{SizeError} = \sum_i \left| s_i^{\text{pred}} - s_i^{\text{gt}} \right|, \qquad s_i^* = \frac{\text{Area}(P_i^*)}{\text{Area}(B^{\text{gt}})},$$

where $P_i$ is the polygon of the $i$-th room and $B^{\text{gt}}$ is the ground-truth apartment boundary. Lower SizeError indicates better agreement with the specified room areas.

**Comparison protocol.** The Condition column of Table 1 specifies the inputs available to each method. We evaluate our model with boundary and graph conditioning (*Ours w/o size*) and with additional target room areas (*Ours*), using guidance scale $w = 3$ for the latter. This comparison does not use window conditioning. Following HouseDiffusion, methods that require per-room point counts receive the ground-truth counts; robustness to unknown counts is evaluated separately in Sec. 4.4.

We do not report GED for GSDiff, since this method does not generate doors and therefore induces a different notion of room adjacency. For compatibility with GSDiff, we additionally report FIDgs and

KIDgs, computed on rasterized floorplans without doors. We also omit SizeError for the boundary-conditioned GSDiff variant and WallPlan, as these methods may generate rooms that are not present in the input graph, making per-room area matching ill-defined.

**Quantitative results.** Table 1 shows that our model conditioned on the boundary and graph already achieves lower FID, KID, and GED than the evaluated baselines, including Graph2Plan under the same set of conditions. Adding target room areas reduces SizeError from 0.163 to 0.028, while further improving FID from 6.43 to 6.05 and GED from 0.33 to 0.28. Boundary agreement also improves, with BoundIoU increasing from 0.966 to 0.974, close to WallPlan's 0.975. Although KID increases slightly with size conditioning, both door-free distribution metrics improve. These results show that the model can use the additional area constraints to improve size compliance while retaining high distribution-level quality and graph consistency.

**Qualitative results.** Fig. 2 complements the quantitative comparison with representative test layouts. The examples illustrate incomplete coverage of the apartment boundary, missing rooms, and connectivity errors in baseline predictions. Our model generally produces cleaner room geometry and follows the prescribed boundary and graph more closely, consistent with the quantitative results. Residual connectivity errors remain in some of our predictions, however, as also reflected by the non-zero GED.

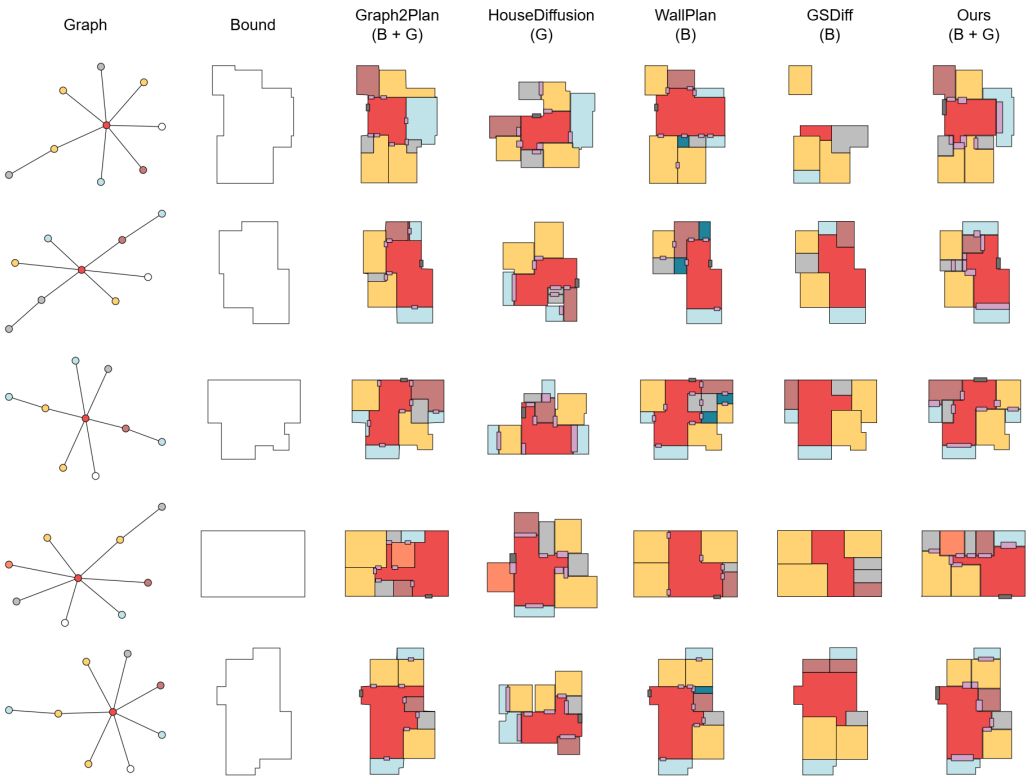


Figure 2: Qualitative comparison on RPLAN test samples under graph and boundary conditioning. *G* and *B* indicate support for graph-adjacency and apartment-boundary conditioning, respectively.

### 4.3 Joint conditioning and controllability

All experiments in this subsection use our framework trained on a version of RPLAN augmented with windows, whereas our model variants in Table 1 are trained on data without windows. Windows are added procedurally as described in Appendix A; entrance doors are taken from the dataset annotations. We evaluate both joint adherence to these additional spatial conditions and control over room areas while the other conditions remain fixed.

Table 2: Quantitative evaluation of window and entrance-door conditioning. *Ours, uncond* predicts windows and the entrance door, while *Ours, cond* uses them as given conditions.

| Method | FID ↓ | KID ↓ | BoundIoU ↑ | GED ↓ | SizeError ↓ | W/D IoU ↑ |
|---|---|---|---|---|---|---|
| Ours, uncond | 6.29 | 0.93 | 0.973 | 0.27 | 0.023 | 0.22 |
| Ours, cond | 5.38 | 0.85 | 0.984 | 0.21 | 0.017 | 0.95 |

Table 3: Room-area control under joint conditioning. The target area of one randomly selected room is multiplied by *Scale*, and the remaining target areas are adjusted proportionally to preserve their total. The graph, boundary, and opening locations remain fixed.

| Scale | FID ↓ | KID ↓ | BoundIoU ↑ | GED ↓ | SizeError ↓ | W/D IoU ↑ |
|---|---|---|---|---|---|---|
| 0.3 | 8.51 | 3.33 | 0.963 | 0.50 | 0.051 | 0.943 |
| 0.5 | 6.98 | 1.84 | 0.968 | 0.42 | 0.040 | 0.949 |
| 1.0 | 5.38 | 0.85 | 0.984 | 0.21 | 0.017 | 0.953 |
| 2.0 | 7.09 | 1.81 | 0.961 | 0.58 | 0.046 | 0.952 |
| 3.0 | 8.84 | 2.76 | 0.952 | 0.76 | 0.092 | 0.951 |

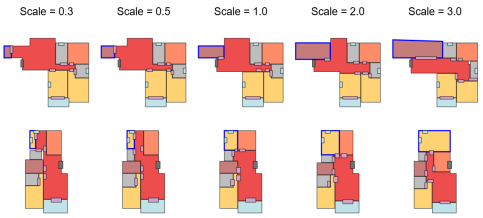


Figure 3: Room-size conditioning under controlled target-area scaling on RPLAN test samples. The blue outline indicates the room whose target area is scaled by *Scale*, demonstrating how the generated layout adapts to the size condition.

**Joint conditioning on openings.** Table 2 evaluates window and entrance-door conditioning in addition to the graph, boundary, and target room areas. The latter three conditions are supplied in both settings: *Ours, uncond* generates windows and the entrance door without conditioning on their locations, whereas *Ours, cond* additionally receives their ground-truth locations. We measure adherence to the prescribed locations using Window/Door IoU (W/D IoU), computed over the generated and ground-truth polygons corresponding to windows and the entrance door. Without explicit opening-location conditioning, the model achieves a W/D IoU of 0.22, consistent with learning opening-placement patterns from the training data. In our experiments, these patterns reflect the procedural window-placement scheme. The same formulation could be trained on real window annotations to learn placement patterns observed in architectural practice. Explicit conditioning increases W/D IoU to 0.95, while also improving other metrics. These results demonstrate joint conditioning on the full set of inputs within our framework.

**Control over room areas.** To evaluate controllability beyond the target areas in the test data, we scale the target area of one randomly selected room in each layout by a factor *Scale*. The remaining target areas are adjusted proportionally so that their sum is unchanged, while the graph, apartment boundary, and window and entrance-door locations are held fixed. Table 3 reports results for factors from 0.3 to 3.0, with SizeError evaluated against the modified targets. Fig. 3 illustrates how the generated room geometry adapts to these changes.

Larger departures from the original areas increase SizeError and GED and worsen the distribution-level metrics, while W/D IoU remains above 0.94 across the tested factors. At a scale of 3.0, for example, SizeError rises from 0.017 to 0.092 and GED from 0.21 to 0.76. The model thus responds to changes in the requested areas under the other spatial and graph conditions, although extreme requests make joint constraint satisfaction more difficult.

The strength of room-area conditioning can additionally be controlled through the classifier-free guidance scale. Increasing the size-guidance scale from $w = 1$ to $w = 3$ reduces SizeError, with a small increase in FID from 5.31 to 5.38. We use $w = 3$ as a compromise between area accuracy and distribution-level quality; the full guidance-scale comparison is provided in Appendix D.

### 4.4 Robustness to the point budget

At inference time, the number of points required to represent each room is generally unknown. We therefore compare an oracle setting, which uses the ground-truth point count for each room, with a fixed-budget setting that assigns $N$ points to each room, $2N$ to the living room, and four points

Table 4: Oracle versus fixed-budget generation. BIoU denotes BoundIoU; Corners counts geometric corners after simplification (GT: 5.28); Match is the percentage of rooms matching the GT corner count. HouseDiffusion is not boundary-conditioned.

| Model | Budget | FID ↓ | BIoU ↑ | GED ↓ | SizeError ↓ | Corners | Match, % ↑ |
|---|---|---|---|---|---|---|---|
| House Diffusion | Oracle | 8.42 | 0.578 | 0.76 | 0.246 | 5.25 | 95.4 |
| | 8 | 20.75 | 0.566 | 1.12 | 0.271 | 6.85 | 12.5 |
| Ours w/o aug | Oracle | 6.31 | 0.969 | 0.31 | 0.160 | 5.14 | 88.7 |
| | 8 | 11.17 | 0.956 | 1.07 | 0.215 | 7.79 | 15.4 |
| Ours | Oracle | 6.43 | 0.966 | 0.33 | 0.163 | 5.08 | 84.4 |
| | 8 | 6.65 | 0.965 | 0.31 | 0.164 | 5.73 | 61.7 |

Table 5: Budget sensitivity of our augmented model. Protocol and column definitions follow Table 4.

| Budget | FID ↓ | GED ↓ | BIoU ↑ | Corners |
|---|---|---|---|---|
| Oracle | 6.43 | 0.33 | 0.966 | 5.08 |
| $N = 4$ | 9.58 | 0.49 | 0.956 | 4.47 |
| $N = 8$ | 6.65 | 0.31 | 0.965 | 5.73 |
| $N = 10$ | 7.20 | 0.42 | 0.963 | 6.19 |
| $N = 12$ | 7.39 | 0.50 | 0.962 | 6.50 |
| $N = 14$ | 8.08 | 0.88 | 0.961 | 7.05 |

to each door. The objective is to maintain generation quality without knowing the individual room counts. These experiments use models trained on data without windows and are performed without room-size conditioning; the reference for our model is the *Ours w/o size* row of Table 1.

**Fixed-budget generation.** Table 4 compares HouseDiffusion and our models trained with and without polygon augmentation under oracle counts and a fixed budget of $N = 8$. Our augmented model largely preserves its oracle generation quality and condition compliance. HouseDiffusion shows a substantial loss of quality, while both HouseDiffusion and our model without augmentation exhibit worse graph consistency and more spurious geometric corners under the fixed budget.

**Effect of augmentation on polygon geometry.** The comparison between our two training variants isolates the contribution of polygon augmentation under the same fixed-budget protocol. At $N = 8$, augmentation improves FID from 11.17 to 6.65 and GED from 1.07 to 0.31. To examine how the models use the allocated points, we also count geometric corners after merging duplicate vertices and removing near-collinear points; the precise procedure is given in Appendix C. The augmented model produces 5.73 corners per room on average, compared with 7.79 without augmentation and 5.28 in the ground truth. The fraction of rooms whose corner count matches the corresponding ground-truth room is also higher, at 61.7% versus 15.4%. This supports the role of augmentation in representing room polygons with redundant points without introducing as many additional geometric corners. Nevertheless, the match rate remains below the augmented model's oracle result of 84.4%, indicating that robustness in the aggregate quality metrics does not imply exact recovery of the ground-truth corner structure. Visual comparisons with and without augmentation under increased point counts are shown in Fig. 5 in Appendix C.

**Sensitivity to the allocated budget.** Table 5 evaluates our augmented model across $N \in \{4, 8, 10, 12, 14\}$. A budget of $N = 4$ leaves insufficient capacity for more complex room shapes, whereas $N = 8$ provides the best overall performance among the tested fixed budgets. Further increases lead to gradual degradation as the model must accommodate more redundant points along existing walls and increasingly produces additional geometric corners. We therefore use $N = 8$ as the default fixed-budget setting. The full sweep for the other models and an additional experiment with randomly increased point counts are provided in Appendix C.

## 5 Conclusion and Limitations

We introduced HouseDiffusion++, a unified diffusion framework for vector floorplan generation jointly conditioned on room adjacency, the apartment boundary, target room areas, and prescribed openings. Our polygon augmentation strategy enables fixed-budget inference with a small gap in generation quality and condition compliance relative to oracle point counts. Experiments on RPLAN demonstrate improved quality and constraint adherence compared with publicly available baselines in the evaluated settings.

Our evaluation is limited to RPLAN with procedurally added windows; generalization to other datasets and learning from real window annotations remain unvalidated. Generated layouts may still contain geometric artifacts or violate some conditions. Optional postprocessing addresses geometric defects, while additional architectural requirements, such as plumbing constraints, remain outside the current formulation.

### AI USE STATEMENT

In this work, we used generative AI tools (large language model assistants) to identify relevant literature and to edit and restructure the manuscript text, including drafting revised passages from author-written notes and checking the manuscript for internal inconsistencies. All AI-assisted text was reviewed and edited by the authors, and all suggested references were checked for accuracy and relevance. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

### ETHICS STATEMENT

This work uses the existing RPLAN dataset and does not involve studies with human participants. The proposed model is intended to assist designers in exploring candidate residential layouts. Generated plans may contain geometric or connectivity errors, and satisfying the modeled constraints does not establish compliance with building regulations, accessibility requirements, or structural safety. Outputs therefore require review and validation by qualified professionals before practical use. Evaluation is limited to RPLAN with procedurally added windows; suitability for other building types and design contexts has not been established.

### REPRODUCIBILITY STATEMENT

The dataset splits, training schedule, optimizer settings, and the augmentation and guidance probabilities are given in Sec. 4.1, together with the evaluation protocol; the fixed-budget protocol is detailed in Appendix C; the window-placement procedure and its parameters in Appendix A; the post-processing implementation in Appendix B; and the architecture of the added components in Appendix E.

## REFERENCES

Laura Carrera, Nil Brullet, Julia Capomaggi, Amadeu Santacana, Ricardo Devesa, Guillem Rosselló, Enrique Romero, and Lluís Ortega. The impact of architecturally qualified data in deep learning methods for the automatic generation of social housing layouts. *Automation in Construction*, 158:105238, 2024. doi: 10.1016/j.autcon.2023.105238.

Jayakrishna Duggempudi, Lu Gao, Ahmed Senouci, Zhe Han, and Yunpeng Zhang. Text-to-layout: A generative workflow for drafting architectural floor plans using llms, 2025. URL https://arxiv.org/abs/2509.00543.

Mohammed Haroon Dupty, Yanfei Dong, Sicong Leng, Guoji Fu, Yong Liang Goh, Wei Lu, and Wee Sun Lee. Constrained layout generation with factor graphs. In *CVPR*, 2024.

Shibo Hong, Xuhong Zhang, Tianyu Du, Sheng Cheng, Xun Wang, and Jianwei Yin. Cons2plan: Vector floorplan generation from various conditions via a learning framework based on conditional diffusion models. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pp. 3248–3256, 2024. doi: 10.1145/3664647.3680681.

Ruizhen Hu, Zeyu Huang, Yuhan Tang, Oliver van Kaick, Hao Zhang, and Hui Huang. Graph2plan: Learning floorplan generation from layout graphs. *ACM TOG*, 39(4):118:1–118:14, 2020. doi: 10.1145/3386569.3392391.

Sizhe Hu, Wenming Wu, Yuntao Wang, Benzhu Xu, and Liping Zheng. Gsdiff: Synthesizing vector floorplans via geometry-enhanced structural graph generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 17323–17332, 2025. doi: 10.1609/aaai.v39i16.33904. URL https://github.com/SizheHu/GSDiff.

Ahti Kalervo, Juha Ylioinas, Markus Häikiö, Antti Karhu, and Juho Kannala. CubiCasa5K: A dataset and an improved multi-task model for floorplan image analysis. In *Image Analysis: 21st Scandinavian Conference, SCIA 2019*, pp. 28–40, 2019. doi: 10.1007/978-3-030-20205-7_3.

Hongjin Lian, Jian Ma, Hongjie Chen, Jia Li, Ruizhen Hu, Yukun Lai, and Kun Li. CG-Floor: Centroid-guided diffusion for large-scale floorplan generation. In *CVPR*, 2026.

Jiachen Liu, Yuan Xue, Jose Duarte, Krishnendra Shekhawat, Zihan Zhou, and Xiaolei Huang. End-to-end graph-constrained vectorized floorplan generation with panoptic refinement. In *ECCV*, pp. 547–562, 2022. doi: 10.1007/978-3-031-19784-0_32.

Jiachen Liu, Yuan Xue, Haomiao Ni, Rui Yu, Zihan Zhou, and Sharon X. Huang. Computer-aided layout generation for building design: A review. *Computational Visual Media*, 2025. doi: 10.26599/CVM.2025.9450484.

Ziniu Luo and Weixin Huang. Floorplangan: Vector residential floorplan adversarial generation. *Automation in Construction*, 142:104470, 2022. doi: 10.1016/j.autcon.2022.104470. URL https://github.com/luozn15/FloorplanGAN.

Ahmed Meselhy and Amal Almalkawi. A review of artificial intelligence methodologies in computational automated generation of high performance floorplans. *npj Clean Energy*, 2025. doi: 10.1038/s44406-025-00002-8.

Fatemeh Mostafavi, Casper van Engelenburg, Seyran Khademi, and Georg Vrachliotis. Floor plan generation: The interplay among data, machine, and designer. *International Journal of Architectural Computing*, 23(2):370–386, 2025. doi: 10.1177/14780771241290649.

Nelson Nauata, Sepidehsadat Hosseini, Kai-Hung Chang, Hang Chu, Chin-Yi Cheng, and Yasutaka Furukawa. House-gan++: Generative adversarial layout refinement network towards intelligent computational agent for professional architects. In *CVPR*, pp. 13632–13641, 2021.

Sizhong Qin, Chengyu He, Qiaoyun Chen, Sen Yang, Wenjie Liao, Yi Gu, and Xinzheng Lu. Chathouse-diffusion: Prompt-guided generation and editing of floor plans, 2024. URL https://github.com/ChatHouseDiffusion/chathousediffusion.

Sizhong Qin, Ramon Elias Weber, and Xinzheng Lu. Tokenization allows multimodal large language models to understand, generate and edit architectural floor plans. In *CVPR*, 2026.

Morteza Rahbar, Mohammadjavad Mahdavinejad, Amir H. D. Markazi, and Mohammadreza Bemanian. Architectural layout design through deep learning and agent-based modeling: A hybrid approach. *Journal of Building Engineering*, 47:103822, 2022. doi: 10.1016/j.jobe.2021.103822.

Mohammad Amin Shabani, Sepidehsadat Hosseini, and Yasutaka Furukawa. Housediffusion: Vector floorplan generation via a diffusion model with discrete and continuous denoising. In *CVPR*, pp. 5466–5475, 2023. URL https://github.com/aminshabani/house_diffusion.

Kaede Shiohara and Toshihiko Yamasaki. Unified vector floorplan generation via markup representation, 2026. URL https://arxiv.org/abs/2604.04859.

Leonardo Stoppani, Davide Bacciu, and Shahab Mokarizadeh. Boundary-constrained diffusion models for floorplan generation: Balancing realism and diversity, 2026. URL https://arxiv.org/abs/2602.01949.

Jiahui Sun, Wenming Wu, Ligang Liu, Wenjie Min, Gaofeng Zhang, and Liping Zheng. Wallplan: Synthesizing floorplans by learning to generate wall graphs. *ACM TOG*, 41(4):1–14, 2022. doi: 10.1145/3528223.3530135.

Abhinav Upadhyay, Alpana Dubey, Veenu Arora, Suma Mani Kuriakose, and Shaurya Agarawal. Flnet: Graph constrained floor layout generation. In *2022 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, pp. 1–6, 2022. doi: 10.1109/ICMEW56448.2022.9859350.

Abhinav Upadhyay, Alpana Dubey, Suma Mani Kuriakose, and Shaurya Agarawal. Floorgan: Generative network for automated floor layout generation. In *Proceedings of the 6th Joint International Conference on Data Science and Management of Data*, pp. 140–148, 2023. doi: 10.1145/3570991.3571057.

Casper van Engelenburg, Fatemeh Mostafavi, Emanuel Kuhn, Yuntae Jeon, Michael Franzen, Matthias Standfest, Jan van Gemert, and Seyran Khademi. MSD: A benchmark dataset for floor plan generation of building complexes. In *ECCV*, volume 15116 of *Lecture Notes in Computer Science*, pp. 60–75, 2024. doi: 10.1007/978-3-031-73636-0_4.

Shidong Wang and Renato Pajarola. Eliminating rasterization: Direct vector floor plan generation with diffplanner. *IEEE TVCG*, 31(10):7906–7922, 2025. doi: 10.1109/TVCG.2025.3559682.

Shidong Wang and Renato Pajarola. Directly from alpha to omega: Controllable end-to-end vector floor plan generation. *IEEE TVCG*, pp. 1–15, 2026. doi: 10.1109/TVCG.2026.3665422. Early access / preprint metadata; update volume and issue after final assignment.

Shidong Wang, Wei Zeng, Xi Chen, Yu Ye, Yu Qiao, and Chi-Wing Fu. Actfloor-gan: Activity-guided adversarial networks for human-centric floorplan design. *IEEE TVCG*, 29(3):1610–1624, 2023. doi: 10.1109/TVCG.2021.3126478.

Ramon Elias Weber, Caitlin Mueller, and Christoph Reinhart. Automated floorplan generation in architectural design: A review of methods and applications. *Automation in Construction*, 140:104385, 2022. doi: 10.1016/j.autcon.2022.104385.

Wenming Wu, Xiao-Ming Fu, Rui Tang, Yuhan Wang, Yu-Hao Qi, and Ligang Liu. Data-driven interior plan generation for residential buildings. *ACM TOG*, 38(6):234:1–234:12, 2019. doi: 10.1145/3355089.3356556.

Biao Xiong, Zhen Peng, Ping Wang, Qiegen Liu, and Xian Zhong. Tlc-plan: A two-level codebook based network for end-to-end vector floorplan generation. *Comput. Graph. Forum*, pp. e70360, 2026. doi: 10.1111/cgf.70360. URL https://github.com/rosolose/TLC-PLAN.

Jun Yin, Pengyu Zeng, Haoyuan Sun, Yuqin Dai, Han Zheng, Miao Zhang, Yachao Zhang, and Shuai Lu. FloorPlan-LLaMa: Aligning architects' feedback and domain knowledge in architectural floor plan generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6640–6662, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.331. URL https://aclanthology.org/2025.acl-long.331/.

Hang Zhang, Anton Savov, and Benjamin Dillenburger. MaskPLAN: Masked generative layout planning from partial input. In *CVPR*, 2024.

Ziyang Zong, Guanying Chen, Zhaohuan Zhan, Fengcheng Yu, and Guang Tan. Housetune: Two-stage floorplan generation with llm assistance. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 14059–14067, 2026. doi: 10.1609/aaai.v40i16.38417.

# APPENDIX

## A WINDOW PLACEMENT

Given an apartment with $N$ room polygons $\{P_i\}_{i=1}^{N}$ and adjacency graph $G$, we compute the outer boundary $B = \bigcup_i \partial P_i$ and sample a target window count $K \sim \mathrm{Uniform}\{1, \ldots, K_{\max}\}$. Rooms are visited in a randomly shifted order to avoid side bias; for each room of eligible type $\mathcal{T}_w$, we sample a per-room budget and place windows on uniformly chosen exterior edges, skipping edges shorter than $s_w$ and removing each used edge to prevent overlap. Each window polygon is appended to the window list $\{W_i\}$ and linked to its host room in $G$. The procedure is summarized in Algorithm 1.

**Algorithm 1** Window placement on a floorplan.

**Input:** $\{P_i\}_{i=1}^{N}$, $G$, $\mathcal{T}_w$, $s_w$, $K_{\max}$.
**Output:** updated $G$ and $\{W_i\}$.
$B \leftarrow$ OUTERCONTOUR$(\{P_i\})$; $K \sim \mathrm{Uniform}\{1, \ldots, K_{\max}\}$; $k \leftarrow 0$
$\pi \leftarrow \mathrm{Roll}\big((1, \ldots, N), \mathrm{Uniform}\{0, \ldots, N-1\}\big)$ ▷ random shift
**for** $i$ in $\pi$ **do**
  **if** $\mathrm{type}(P_i) \notin \mathcal{T}_w$ **or** $k \geq K$ **then continue**
  **end if**
  $n_i \sim \mathrm{Uniform}\{1, \ldots, K\}$; $\mathcal{E}_i \leftarrow$ FIND_EXTERIOR_EDGES$(P_i, B)$
  **for** $t = 1, \ldots, n_i$ **do**
    **if** $\mathcal{E}_i = \varnothing$ **or** $k \geq K$ **then break**
    **end if**
    sample $e = (B_i, B_j)$ uniformly from $\mathcal{E}_i$
    **if** $\|B_j - B_i\| < s_w$ **then**
      $\mathcal{E}_i \leftarrow \mathcal{E}_i \setminus \{e\}$; **continue**
    **end if**
    construct $W$ from $(e, s_w)$ via Eq. (1)
    append $W$ to $\{W_i\}$; link $W$ to $P_i$ in $G$; $k \leftarrow k + 1$; $\mathcal{E}_i \leftarrow \mathcal{E}_i \setminus \{e\}$
  **end for**
**end for**
**return**

For a chosen edge $e = (B_i, B_j)$ with unit tangent $\mathbf{u} = (B_j - B_i)/\|B_j - B_i\|$, inward normal $\mathbf{n}$, center $c = \frac{1}{2}(B_i + B_j)$, and size $s_w$, the window corners $W_j = \{w_{j,1}, \ldots, w_{j,4}\} \subset \mathbb{R}^2$ are

$$\begin{aligned} w_{j,1} &= c - \tfrac{s_w}{2}\mathbf{u}, \qquad w_{j,2} = c + \tfrac{s_w}{2}\mathbf{u}, \\ w_{j,3} &= w_{j,2} - \tfrac{s_w}{2}\mathbf{n}, \quad w_{j,4} = w_{j,1} - \tfrac{s_w}{2}\mathbf{n}. \end{aligned} \tag{1}$$

We use $\mathcal{T}_w = \{\text{kitchen, bedroom, bathroom, dining room, study}\}$, $s_w = 14/256$, and $K_{\max} = 4$. The subroutine FIND_EXTERIOR_EDGES returns edges of $P_i$ whose intersection with $\partial B$ recovers the full edge length.

## B POSTPROCESSING DETAILS

This appendix describes the optional deterministic post-processing procedure used to correct geometric artifacts in generated vector floorplans before further use in architectural software. Let $S$ denote the apartment shell and $\{R_i\}_{i=1}^{n}$ the generated room polygons. The desired output consists of valid polygons that remain inside the shell, do not overlap, and together cover the apartment:

$$R_i \subseteq S, \qquad \mathrm{int}(R_i) \cap \mathrm{int}(R_j) = \varnothing \quad (i \neq j), \qquad S = \bigcup_{i=1}^{n} R_i.$$

**Overview.** We first repair invalid polygons and clip them to the apartment shell. Snapped coordinates from the shell and room boundaries define a grid whose cells are assigned to individual rooms. Confident assignments follow the overlap with the generated polygons; ambiguous or uncovered cells

are assigned using boundary contact, distance, perimeter change, and remaining target-area demand. Cells assigned to the same room are then merged into polygons. The following paragraphs specify the geometric discretization, ownership initialization, scoring rule, and fallback cases. Complete coverage requires that the cell partition cover the shell and that all assigned cells be retained in the reconstructed polygons; the cell filtering, component selection, and fallback rules below delimit this property.

**Polygon normalization and validity correction.** All input contours are first converted to planar polygonal geometries. If a room polygon is explicitly closed by repeating the first vertex, the repeated endpoint is removed before constructing the polygon. Degenerate contours with fewer than three vertices are discarded. We use a validity correction operator

$$\mathrm{Repair}(P) = \begin{cases} P, & \text{if } P \text{ is empty or already valid,} \\ \mathrm{buffer}(P, 0), & \text{otherwise,} \end{cases}$$

which removes common degeneracies such as self-intersections and zero-area fragments before boolean operations. If a boolean operation produces several disconnected polygonal components, we keep the largest non-empty component. This choice makes the procedure conservative: small spurious fragments are suppressed, while the dominant predicted room geometry is preserved.

**Coordinate snapping.** Let $\widetilde{R}_i = \mathrm{Repair}(R_i) \cap S$ be the clipped room geometry. We collect all horizontal and vertical grid coordinates from the shell boundary and the clipped rooms,

$$X_0 = \{x : (x, y) \in \partial S \cup \bigcup_i \partial \widetilde{R}_i\}, \qquad Y_0 = \{y : (x, y) \in \partial S \cup \bigcup_i \partial \widetilde{R}_i\}.$$

The values in each set are sorted and grouped whenever they differ by at most $\varepsilon_{\mathrm{snap}}$ from the current group mean. Each group is then replaced by its median value. This step merges nearly coincident wall lines produced by small numerical errors, while keeping the global coordinate structure induced by the model prediction. The shell bounding-box coordinates are always inserted back into the snapped sets to avoid losing the exterior extent of the apartment.

**Cell construction.** Given the snapped coordinate sets $X = \{x_1 < \cdots < x_p\}$ and $Y = \{y_1 < \cdots < y_q\}$, we form candidate rectangles

$$Q_{ab} = [x_a, x_{a+1}] \times [y_b, y_{b+1}], \qquad 1 \le a < p, \quad 1 \le b < q.$$

Each rectangle is intersected with the apartment shell,

$$C_{ab} = \mathrm{Repair}(Q_{ab} \cap S),$$

and is kept only if it has positive area and lies sufficiently inside the shell. In practice, a cell is accepted if either its center belongs to $S$ or the area ratio $|C_{ab}|/|Q_{ab}|$ exceeds a threshold $\tau_{\mathrm{in}}$. This avoids adding irrelevant exterior slivers near the boundary.

**Ownership initialization.** The initial owner of a cell is determined by maximum overlap with clipped room polygons. A cell $C$ is assigned to room $i$ only if the overlap ratio is sufficiently large,

$$\frac{|C \cap \widetilde{R}_i|}{|C|} \ge \tau_{\mathrm{own}}.$$

Otherwise, it remains unassigned. This distinction is important: confidently predicted regions are preserved, while uncovered regions and overlap-conflict regions are deferred to the scoring stage.

**Target-area estimation.** If all rooms have valid explicit target areas, these values are normalized to the shell area,

$$A_i^\star = \frac{a_i}{\sum_j a_j} |S|.$$

If explicit areas are unavailable, we use the current proportions induced by confidently owned cells. This fallback prevents the post-processing step from inventing new room-size preferences when no size signal is provided.

**Algorithm 2** Detailed grid-cell postprocessing

**Input:** apartment shell $S$, room polygons $\{R_i\}_{i=1}^n$, snapping threshold $\varepsilon_{\text{snap}}$, ownership threshold $\tau_{\text{own}}$, inside threshold $\tau_{\text{in}}$.
**Output:** corrected room polygons $\{R_i^{\text{new}}\}_{i=1}^n$.
$S \leftarrow \text{LargestPolygon}(\text{Repair}(S))$.
**for** $i = 1, \dots, n$ **do**
    $\widetilde{R}_i \leftarrow \text{LargestPolygon}(\text{Repair}(R_i) \cap S)$.
    **if** $\widetilde{R}_i$ is empty **then**
        discard room $i$ from the post-processing assignment.
    **end if**
**end for**
Collect coordinate sets $X_0, Y_0$ from $\partial S$ and $\{\partial \widetilde{R}_i\}_{i=1}^n$.
$X, Y \leftarrow \text{Snap}(X_0, Y_0; \varepsilon_{\text{snap}})$.
Insert the shell bounding-box coordinates into $X$ and $Y$.
$\mathcal{C} \leftarrow \varnothing$.
**for** each consecutive pair $(x_a, x_{a+1})$ in $X$ **do**
    **for** each consecutive pair $(y_b, y_{b+1})$ in $Y$ **do**
        $Q \leftarrow [x_a, x_{a+1}] \times [y_b, y_{b+1}]$.
        $C \leftarrow \text{LargestPolygon}(\text{Repair}(Q \cap S))$.
        **if** $C$ has positive area and is sufficiently inside $S$ **then**
            add $C$ to $\mathcal{C}$.
        **end if**
    **end for**
**end for**
**if** $|\mathcal{C}| = 0$ or $|\mathcal{C}| > N_{\max}$ **then**
    **return** the original room polygons.
**end if**
**for** each cell $C \in \mathcal{C}$ **do**
    $j^\star \leftarrow \arg\max_j |C \cap \widetilde{R}_j|$.
    **if** $|C \cap \widetilde{R}_{j^\star}|/|C| \geq \tau_{\text{own}}$ **then**
        $o(C) \leftarrow j^\star$.
    **else**
        $o(C) \leftarrow -1$.
    **end if**
**end for**
Estimate target areas $A_i^\star$ from explicit room areas when available; otherwise use current cell-area proportions.
Sort all cells with $o(C) = -1$ by distance to $\bigcup_i \widetilde{R}_i$.
**for** each unassigned cell $C$ in sorted order **do**
    Compute current room unions $U_i = \bigcup_{C':\, o(C')=i} C'$ and current areas $A_i = |U_i|$.
    $o(C) \leftarrow \arg\max_i \text{Score}(C, i)$.
**end for**
**for** $i = 1, \dots, n$ **do**
    $R_i^{\text{new}} \leftarrow \text{LargestPolygon}\left(\text{Repair}\left(\bigcup_{C:\, o(C)=i} C\right) \cap S\right)$.
    remove redundant collinear vertices from $R_i^{\text{new}}$.
**end for**

**Cell assignment score.** For an unassigned cell $C$ and a candidate room $i$, let $U_i$ be the current union of cells assigned to room $i$ and let $A_i = |U_i|$. We use

$$\begin{aligned}\text{Score}(C, i) = \lambda_c\ &\text{length}(\partial C \cap \partial U_i) - \lambda_d\ \text{dist}(C, U_i) \\ &- \lambda_p\,[\text{perim}(U_i \cup C) - \text{perim}(U_i)]_+ + \lambda_a \frac{A_i^\star - A_i}{|S|}|C|.\end{aligned}$$

The first term favors cells that share a boundary with the candidate room. The second term discourages assignments across distant parts of the layout. The third term penalizes additions that make the room boundary more fragmented. The last term biases growth toward rooms that are currently smaller than their target area. In our implementation, empty candidate rooms are ignored during scoring because they provide no geometric support for local growth.

Table 6: Effect of postprocessing on our model with graph, boundary, and room-size conditioning. Metrics and evaluation protocol follow Table 1.

| Method | FID ↓ | FIDgs ↓ | KID ↓ | KIDgs ↓ | BoundIoU ↑ | GED ↓ | SizeError ↓ |
|---|---|---|---|---|---|---|---|
| Ours | 6.05 | 3.98 | 0.96 | 0.52 | 0.974 | 0.28 | 0.028 |
| Ours + pp | 7.51 | 4.49 | 1.24 | 1.05 | 1.000 | 0.28 | 0.030 |

**Ordering of ambiguous cells.** Unassigned cells are processed from near to far with respect to the initial union of clipped rooms. This makes the assignment propagate outward from reliable geometry rather than filling large empty areas arbitrarily. After each accepted assignment, the corresponding room union is updated, so later cells can attach to the newly expanded room geometry.

**Safeguards.** The procedure is skipped and the original prediction is returned in several degenerate cases: empty shell, no valid room polygons, no valid cells, or an excessive number of generated cells. The last condition avoids unstable behavior on overly fragmented coordinate grids. After reconstruction, redundant collinear vertices are removed to keep the final polygon representation compact.

**Effect of the procedure.** The post-processing stage addresses four common failure modes of vector generation. First, clipping removes geometry outside the apartment shell. Second, assigning a unique owner to each cell resolves room overlaps. Third, unassigned cells fill holes and uncovered regions inside the shell. Fourth, coordinate snapping and cell-based reconstruction reduce small wall misalignments and suppress unnecessary diagonal fragments. Since all corrections are made using the grid induced by the predicted layout itself, the procedure changes uncertain regions while preserving the stable geometric structure of the generated sample.

### B.1 Quantitative and qualitative evaluation

Table 6 compares our model before and after postprocessing under the protocol of Table 1. Post-processing increases BoundIoU from 0.974 to 1.000, while GED remains unchanged and SizeError increases slightly from 0.028 to 0.030. Both FID and KID increase, indicating that improved boundary coverage does not translate into better distribution-level scores. We therefore report these results separately from the main comparison of generative models.

Fig. 4 illustrates the effect on selected challenging samples whose raw predictions contain visible geometric errors. The before-and-after comparisons show how the procedure corrects local artifacts while retaining the overall room arrangement.

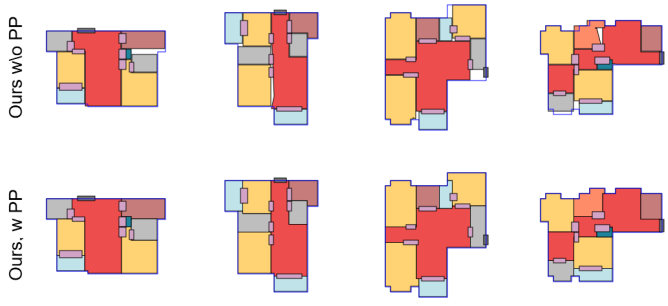


Figure 4: Selected challenging layouts before and after postprocessing.

# C Additional point-budget experiments

## C.1 Full fixed-budget sweep

Table 7: Full fixed-budget sweep (raw outputs, no size conditioning, same pipeline, full-corner ground truth, and test ids as Table 1). Every room receives $N$ points, the living room $2N$, and doors four. HouseDiffusion is not boundary-conditioned, hence its low BoundIoU. Ground-truth rooms have 5.28 geometric corners on average. KID is not reported for the model trained without augmentation.

| Model | $N$ | FID ↓ | KID ↓ | BoundIoU ↑ | GED ↓ | SizeError ↓ | Corners | Match, % ↑ |
|---|---|---|---|---|---|---|---|---|
| HouseDiffusion | Oracle | 8.42 | 1.92 | 0.578 | 0.76 | 0.246 | 5.25 | 95.4 |
| | 4 | 14.39 | 8.88 | 0.566 | 0.96 | 0.273 | 4.29 | 69.5 |
| | 8 | 20.75 | 18.65 | 0.566 | 1.12 | 0.271 | 6.85 | 12.5 |
| | 10 | 20.15 | 19.18 | 0.567 | 1.51 | 0.267 | 8.57 | 2.6 |
| | 12 | 66.47 | 84.32 | 0.551 | 1.88 | 0.291 | 9.96 | 0.7 |
| | 14 | 37.84 | 44.16 | 0.579 | 2.01 | 0.279 | 11.33 | 0.4 |
| Ours w/o aug | Oracle | 6.31 | 0.85 | 0.969 | 0.31 | 0.160 | 5.14 | 88.7 |
| | 4 | 11.00 | 7.31 | 0.960 | 0.75 | 0.207 | 4.49 | 71.6 |
| | 8 | 11.17 | 10.68 | 0.956 | 1.07 | 0.215 | 7.79 | 15.4 |
| | 10 | 13.61 | 16.72 | 0.952 | 1.55 | 0.232 | 9.32 | 6.1 |
| | 12 | 22.19 | 25.11 | 0.932 | 2.03 | 0.256 | 10.82 | 2.6 |
| | 14 | 36.58 | 39.04 | 0.909 | 2.79 | 0.301 | 12.48 | 1.0 |
| Ours | Oracle | 6.43 | 0.89 | 0.966 | 0.33 | 0.163 | 5.08 | 84.4 |
| | 4 | 9.58 | 4.46 | 0.956 | 0.49 | 0.190 | 4.47 | 71.5 |
| | 8 | 6.65 | 1.03 | 0.965 | 0.31 | 0.164 | 5.73 | 61.7 |
| | 10 | 7.20 | 1.89 | 0.963 | 0.42 | 0.168 | 6.19 | 52.8 |
| | 12 | 7.39 | 1.92 | 0.962 | 0.50 | 0.172 | 6.50 | 45.6 |
| | 14 | 8.08 | 2.59 | 0.961 | 0.88 | 0.179 | 7.05 | 35.9 |

**Protocol.** All rows are raw outputs without post-processing and without size conditioning, so the reference for our model is the *Ours w/o size* row of Table 1. The corner-token budget is fixed per room ($N$ points, $2N$ for the living room, four for doors). Geometric corners are counted after merging duplicate vertices and removing near-collinear ones, where a vertex is near-collinear if it deviates by less than 1 px of the 256-px raster from the segment joining its neighbors. *Match* is the fraction of rooms whose geometric corner count equals that of the corresponding ground-truth room. Ground-truth rooms have 5.28 geometric corners on average (69% have four and 85% at most six) but store 5.92 vertices, i.e., the ground truth itself contains redundant collinear vertices. The same protocol is applied to the original HouseDiffusion and to our model trained without augmentation. We report the corner statistics on raw outputs only: post-processing removes redundant collinear vertices by construction (Appendix B), so the same statistics after post-processing would measure the repair step rather than the model. We do not report wall-count error after simplification.

**Results.** Table 7 extends Table 4 to all budgets. For our model, degradation with $N$ is gradual: FID stays between 6.65 and 8.08 for $N = 8$–$14$, and the mean geometric corner count grows from 5.73 to 7.05. Without augmentation, surplus points increasingly become geometric corners (7.79 at $N = 8$, 12.48 at $N = 14$) and the metrics collapse in the same way as for the original HouseDiffusion. The gap between the two variants of our model is therefore attributable to the training-time augmentation, not to the inference-time budget, whose only role is to provide representational slack for rooms whose corner count is unknown.

## C.2 Randomly increased point counts

This experiment uses models trained on RPLAN augmented with windows, with graph, boundary, room-size, and opening-location conditioning enabled, as in Sec. 4.3. It therefore complements the fixed-budget sweep under a different conditioning setup. The increased counts are constructed relative to ground-truth counts, so this experiment tests robustness to redundant points rather than inference without access to individual room counts.

In Sec. 3.4, we introduced a polygon augmentation strategy to reduce the train–inference mismatch caused by the unknown number of corners at test time. To evaluate its effect, we train an additional

Table 8: Effect of polygon augmentation under joint conditioning. Models trained with and without augmentation are evaluated using the original point counts and randomly increased counts (More Points).

| Test Data | Method | FID ↓ | KID ↓ | BoundIoU ↑ | GED ↓ | SizeError ↓ | W/D IoU ↑ |
|---|---|---|---|---|---|---|---|
| Original | Ours, w aug | 5.38 | 0.85 | **0.984** | **0.21** | **0.017** | **0.953** |
| | Ours, w/o aug | **4.71** | **0.31** | 0.983 | **0.21** | **0.017** | 0.924 |
| More Points | Ours, w aug | **5.41** | **0.92** | **0.984** | **0.22** | **0.017** | **0.953** |
| | Ours, w/o aug | 16.23 | 11.87 | 0.947 | 0.41 | 0.051 | 0.926 |

Table 9: Effect of the classifier-free guidance scale $w$ for room-size conditioning.

| CFG scale | FID ↓ | KID ↓ | GED ↓ | SizeError ↓ |
|---|---|---|---|---|
| $w = 1$ | **5.31** | **0.68** | **0.211** | 0.023 |
| $w = 2$ | 5.39 | 0.78 | 0.228 | 0.018 |
| $w = 3$ | 5.38 | 0.85 | 0.213 | **0.017** |
| $w = 4$ | 5.52 | 0.84 | 0.230 | **0.017** |

variant of our model on data without augmentations. We compare both models on a modified test set, where each room polygon is assigned a random even number of additional points compared to the ground-truth polygon. The quantitative results are reported in Table 8. Without the proposed augmentation, the model often interprets redundant points as new geometric corners, leading to spurious edges and distorted room shapes. In contrast, the augmented model is substantially more robust and tends to place extra points along existing walls, preserving the intended room geometry. Fig. 5 provides visual comparisons of the two models under increased point counts.

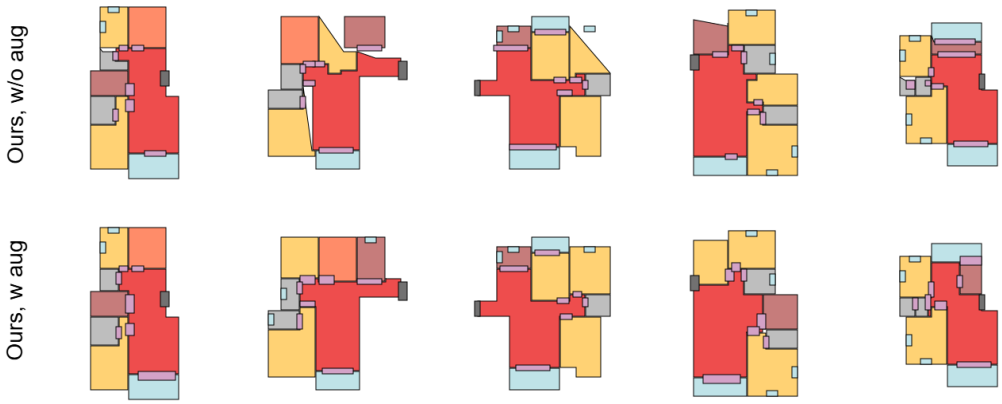


Figure 5: Effect of polygon augmentation on RPLAN test samples with additional boundary points. Here, we compare models, trained with augmentations and without.

# D Additional conditioning experiments

## D.1 Classifier-free guidance for room-size conditioning

We evaluate the guidance scale $w$ using the model trained on window-augmented RPLAN and conditioned jointly on the graph, boundary, target room areas, and window and entrance-door locations, following Sec. 4.3. Table 9 shows that increasing $w$ from 1 to 3 improves SizeError from 0.023 to 0.017, while FID increases slightly from 5.31 to 5.38. Increasing $w$ further to 4 provides no additional improvement in SizeError at the reported precision and increases FID and GED. We therefore use $w = 3$ as the default guidance scale.

## E MODEL ARCHITECTURE

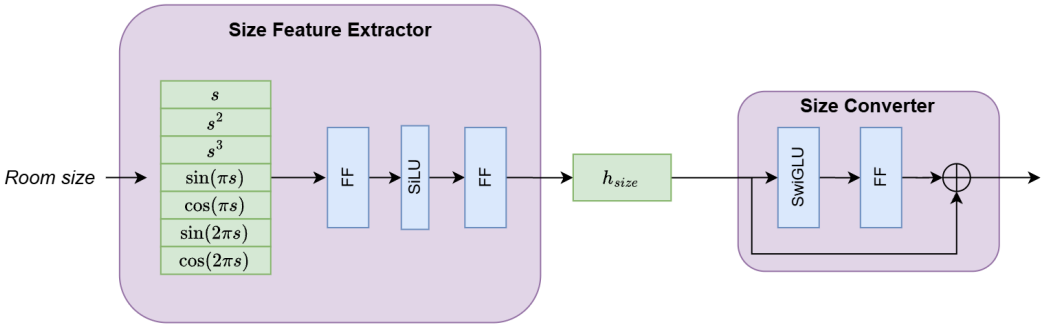


Figure 6: Architecture of Size Feature Extractor and Size Converter used to extract size embeddings.

Our method introduces three additional architectural components on top of the original denoising Transformer: room-size conditioning, spatial-feature encoding, and an additional attention mechanism for injecting spatial constraints. The Transformer blocks used for the new attention module follow the same design as the attention blocks in the original architecture. The Spatial Encoder is built from the same block design as well, but uses only a single attention type. We use the same number of Spatial Encoder blocks as the number of Transformer blocks in the denoising network.

The architecture of the Size Feature Extractor is shown in Fig. 6. Given a target room-area ratio $s_i \in [0, 1]$, we first normalize it as $a_i = 2s_i - 1$ and use a fixed seven-dimensional feature basis combining low-order polynomial terms and sinusoidal features:

$$\phi(a_i) = \left[a_i,\ a_i^2,\ a_i^3,\ \sin(\pi a_i),\ \cos(\pi a_i),\ \sin(2\pi a_i),\ \cos(2\pi a_i)\right].$$

These features are concatenated along the hidden dimension and then passed through two fully connected layers to obtain the room-size embedding $h_{\text{size}}$.

For each Transformer layer, we introduce a separate Size Converter module, whose architecture is shown in Fig. 6. The Size Converter maps the shared size embedding $h_{\text{size}}$ to a layer-specific representation $\hat{h}_{\text{size}}$, which is then added to the main token features before the corresponding attention block. The module follows a SwiGLU-style gated feed-forward design:

$$\hat{h}_{\text{size}} = h_{\text{size}} + W_o\left(\text{SiLU}(W_1 h_{\text{size}}) \odot W_2 h_{\text{size}}\right),$$

where $W_1$ and $W_2$ project the input to an intermediate hidden dimension, $W_o$ projects it back to the model dimension, and $\odot$ denotes element-wise multiplication. We initialize $W_o$ with zeros, so that each Size Converter initially behaves as an identity mapping.

**Room-area encoding alternatives.** To assess sensitivity to the choice of area encoding, we compare our fixed feature basis with a scalar MLP and a Fourier encoding, keeping the downstream conditioning pathway and the training and evaluation protocols unchanged. Table 10 reports the results under graph, boundary, and room-area conditioning. All three variants achieve similar room-area accuracy, with SizeError between 0.028 and 0.030, while no encoding performs best across all reported metrics. These results suggest that accurate room-area conditioning does not depend strongly on the particular feature basis used in our implementation.

Table 10: Room-area encoding alternatives under the protocol of the *Ours* row in Table 1: B+G+S conditioning, ground-truth point counts, $w = 3$, no opening-location conditioning or post-processing.

| Area encoding | FID ↓ | KID ↓ | BoundIoU ↑ | GED ↓ | SizeError ↓ |
|---|---|---|---|---|---|
| Fixed basis (ours) | 6.05 | 0.96 | 0.974 | 0.28 | 0.028 |
| Scalar MLP | 6.13 | 0.93 | 0.975 | 0.24 | 0.028 |
| Fourier | 5.96 | 0.64 | 0.974 | 0.27 | 0.030 |