GENERATION OF HIGH-LEVEL CONCEPTS IN 3D SCENE GRAPHS VIA AUTOREGRESSIVE DIFFUSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Indoor 3D Scene Graphs (3DSGs) enhance robot-observed geometric primitives, such as wall planes, with hierarchies of increasingly abstract metric-semantic concepts (walls, rooms, floors, buildings), each represented as a node with a semantic label, a 3D position, and links to the lower-level elements it groups. These hierarchies support robotic perception and Simultaneous Localization and Mapping (SLAM), yet inferring them automatically from the observed primitives remains an open problem: classical methods rely on rules hand-crafted for each concept, while existing learning-based methods use separate models for structure and for geometry, and do not generate beyond walls and rooms. We propose a unified graph generative model based on autoregressive diffusion that grows a complete 3DSG bottom-up from the observed planes: at each level, it jointly samples how many concept nodes to add, their labels and positions, and their connections to the level below, without any concept-specific module. To evaluate generated hierarchies, we adapt the Fused Gromov–Wasserstein distance to match generated and ground-truth nodes on both topology and features. Across synthetic scenes, real architectural floor plans, and robotic sensor data, up to city level, our model outperforms prior learning-based baselines and, on all the datasets but one, improves on a one-shot diffusion model even when the latter is given the ground-truth graph size, with the largest gains on the deepest hierarchies and on robot-recorded environments.

1 INTRODUCTION

Robots operating in complex indoor environments must reason about spatial structure at multiple scales, from individual sensor measurements to building-wide topology. 3D Scene Graphs (3DSGs) in robotics (Armeni et al., 2019; Hughes et al., 2022) have emerged as a compact representation of this structure, combining semantic, geometric, and relational information. As illustrated in Fig. 1, the hierarchy comprises (i) an observed, low-level layer of planes measured directly from sensor data, and (ii) higher-level layers that aggregate lower-level elements into increasingly abstract topological concepts (e.g., rooms). Formally, a 3DSG is a graph: every plane and every concept is a node carrying a semantic label and a 3D centroid, and edges encode part-of relations that link each concept to the lower-level nodes it aggregates. These concepts, together with their estimated centroids, enhance downstream robotic applications such as Simultaneous Localization And Mapping (SLAM) (Bavle et al., 2022), global localization (Shaheer et al., 2025), path planning (Ejaz et al., 2025), and multi-robot communication (Fernandez-Cortizas et al., 2024).

Despite extensive work on predicting relations between observed nodes (Wu et al., 2021), inferring complete multi-level hierarchies from low-level observations remains largely open. Classical pipelines hand-craft one algorithm per concept, mostly rooms (Luperto et al., 2022; Hughes et al., 2022), while existing learning-based methods stop generation at walls and rooms, split structure and geometry across separate models, or are restricted in layout complexity (Millan-Romera et al., 2026).

To overcome these limitations we propose a unified model based on Deep Graph Generation (DGG) (Cognolato et al., 2026b; Vignac et al., 2023b) that generates complete, multi-level 3DSGs directly from the graph of observed planes, without ad-hoc concept definitions. The model learns a distribution over hierarchical scene graphs and, at inference, extends the observed graph bottom-up: at each level it adds the new concept nodes together with their labels, centroids, and edges to the

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.

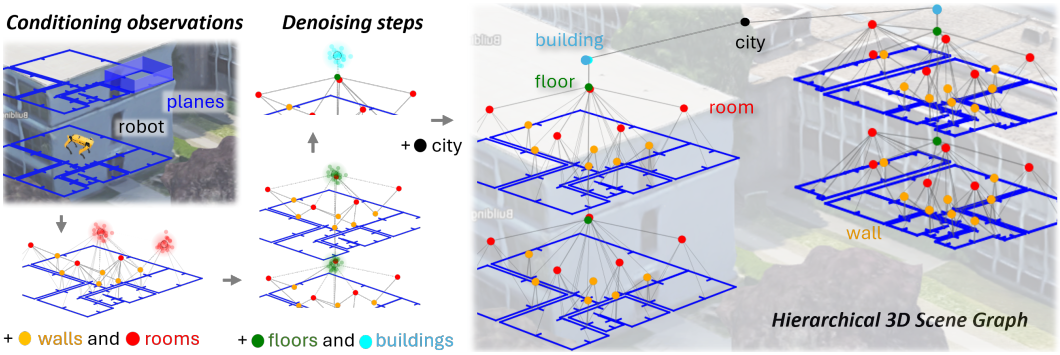

Figure 1: Hierarchical 3D Scene Graph Generation. Starting from wall planes observed by the robot (blue lines), levels are generated bottom-up, with nodes colored by class and placed at their centroids. Grey edges denote part-of relations.

level below, conditioned on the levels already generated. This yields consistent hierarchies spanning walls, rooms, floors, buildings, and cities. We consider a static environment and an offline setting: the input is the final set of planes mapped by the robot, and the full hierarchy is generated from it once. To evaluate the resulting graphs holistically, we adapt the Fused Gromov-Wasserstein (FGW) metric (Vayer et al., 2020) to jointly account for semantic, metric, and relational fidelity.

Our key contributions are as follows: (1) To the best of our knowledge, we present the first autoregressive, diffusion-based DGG model for 3DSG completion, generating multi-level hierarchies conditioned on graphs of robot-observed planes. (2) We introduce an adaptation of FGW to evaluate hierarchical 3DSG generation via structure–attribute graph matching, jointly capturing relational, semantic, and metric consistency. (3) We evaluate on diverse synthetic and real-world scenarios, showing higher accuracy and broader hierarchical coverage than prior methods, and we publicly release the datasets and code to facilitate benchmarking and future research.

2 Related Work

Graph generation. The generation of structured data in the form of graphs has been applied to drug design (Zang & Wang, 2020), material design (Maziarka et al., 2020), and social network analysis (Grover et al., 2019). The goal of graph generation is to approximate an unknown data graph distribution $q(G)$, of which a dataset is available, with a learned model $p_\theta(G)$, and sample new graphs from it with the same characteristics. State of the art methods include Diffusion Models (Vignac et al., 2023a) and Autoregressive Models (Kong et al., 2023). There have been efforts to extend graph generation to multimodal graphs, containing both discrete labels and 3D positions. For 3D attributed graphs, methods must ensure SE(3) invariance, i.e., the probability distribution should be invariant to rotations and translations. This has been addressed through invariant features (Vignac et al., 2023b) or distance-based generation (Cognolato et al., 2026a). The same invariance must be carried over for 3DSG generation, but these methods are not designed for extending existing graphs. The solution adopted in this paper is to apply graph autoregressive models, generating graphs as a sequence of node and edge insertions, incorporating the generation of 3D positions, labels and edges in each step.

Generation of high-level concepts in 3DSGs. Classical 3DSG pipelines rely on concept-specific heuristics tied to a sensor modality: room segmentation from occupancy maps (Luperto et al., 2022) or free space clustering (Hughes et al., 2022); and floor detection via geometric cues (Bavle et al., 2025). While effective, these pipelines typically encode concept-specific heuristics that are difficult to reuse across different high-level concepts. The first learning-based approach considering these properties at the graph level was Neural Trees (Talak et al., 2021), which performs node classification via message passing on an auxiliary tree whose nodes correspond to subgraphs of the original graph, enabling hierarchical aggregation of set-level structure. Other works focused only on the definition of relations between observed objects (Wu et al., 2021; Yang et al., 2017). For node generative approaches, Millan-Romera et al. (2024) introduce learned high-level node generation by classifying

semantic relations between plane primitives to infer walls and rooms, but require separate Graph Attention Network (GAT) models per concept type, assume rectangular layouts, and do not predict geometric attributes (centroids) for generated nodes, which are instead computed by a hand-crafted function from the grouped planes. Millan-Romera et al. (2026) unify edge classification into a single model and add centroid estimation via an additional Graph Neural Network (GNN), handling non-rectangular rooms but requiring every wall to consist of exactly two planes. However, both methods share fundamental limitations: they depend on intermediate edge classification followed by classical community detection algorithms, employ separate architectures for structural and geometric inference, and generate only two node types (walls and rooms), preventing extension to multi-level hierarchies. In contrast, our unified model jointly generates structure and centroids across arbitrary depths without concept-specific modules.

3 PROBLEM DEFINITION

We represent sets with calligraphic letters ($\mathcal{S}$); scalars with lower case letters (s); vectors with lower case bold letters ($\boldsymbol{v}$); matrices and tensors with capital bold letters ($\boldsymbol{M}$). We define graphs as tuples $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, \dots, v_n\}$ is the set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. We denote functions on nodes as $f : \mathcal{V} \to \mathcal{D}$, where $\mathcal{D}$ is the codomain, e.g., the set of labels in case f is a node labeling function. Given a 3D node centroid function $\rho : \mathcal{V} \to \mathbb{R}^3$, we define the node 3D Euclidean distance $d : \mathcal{V} \times \mathcal{V} \to \mathbb{R}_+$ as $d(v, v') = \|\rho(v) - \rho(v')\|$ where $\|\cdot\|$ is the Euclidean norm.

We represent the 3D structure of a man-made indoor environment, e.g., a building, as a set $\mathcal{P}$ of 3D vertical planes, each of the form $\boldsymbol{\pi} = (\boldsymbol{n}, l, \boldsymbol{r})$, where $\boldsymbol{n} \in \mathbb{R}^3$ is the 3D normal orientation, $l \in \mathbb{R}_+$ is its non-negative width and $\boldsymbol{r} \in \mathbb{R}^3$ is the 3D centroid of the plane (Millan-Romera et al., 2024). These planes are typically detected by the robot using a LiDAR sensor or RGB-D camera, employing SLAM (S-Graphs method (Bavle et al., 2022)) to filter outliers and refine plane estimates. The hierarchical 3DSG is defined bottom-up in the following order: *plane*, *wall*, *room*, *floor*, *building*, *city*, where most of the layers present a tree-like structure. Each high-level concept node directly depends on its children in the layer below. The exception is the wall layer, which connects to planes but has no parent node, breaking the strict hierarchy, a design choice consistent with other works in the field (Millan-Romera et al., 2026). 3DSGs are undirected labeled graphs $G = (\mathcal{G}, \lambda, \rho, \pi)$, where $\mathcal{G}$ is the undirected graph topology; $\lambda : \mathcal{V} \to \mathcal{C}$ is the function labeling nodes with their respective concepts $c \in \mathcal{C}$; $\rho : \mathcal{V} \to \mathbb{R}^3$ is the 3D centroid function, returning coordinates $\boldsymbol{r}$ for each node; $\pi : \mathcal{V} \to \mathbb{R}^3 \times \mathbb{R}_+$ is the plane features function, returning the plane normal vector and width $(\boldsymbol{n}, l)$ for plane nodes, and a vector of zeroes otherwise. We define 3DSG generation as the task of inferring the correct 3DSG G with high-level concepts given the set of all planes $\mathcal{P}$ observed by the robot as evidence. Formally, we seek to train a conditional generative model $p_\theta(G|\mathcal{P})$ that approximates the dataset distribution $q(G|\mathcal{P})$. We assume a static environment and an offline setting: $\mathcal{P}$ is the final, converged set of planes produced by the SLAM back-end after mapping is complete, and the 3DSG is generated once from it. Neither the scene nor the observations change during generation.

4 GENERATING 3DSGS

We model the hierarchy generation as the process of growing a graph, repeatedly adding new nodes and edges to the starting planes. The overall architecture is shown in Fig. 2. First, we transform the input set of 3D planes $\mathcal{P}$ into nodes, adding auxiliary edges based on their proximity (see Section 4.1). Then, the learned generative process proceeds iteratively: at each iteration, new nodes are introduced, after which their labels λ in $\mathcal{C}$ (e.g., room, floor), connectivity $\mathcal{E}$, and positions ρ are sampled. The model then decides whether to halt or continue with a new iteration. The growing process is implemented by the FLexible Autoregressive Graph Generation (FLAGG) framework (Cognolato et al., 2026b), while the connectivity and attributes are generated by a one-shot model inside FLAGG; Fig. 2 depicts the pipeline with the MiDi-based Fill module (OURS). We refer to Sec. 4.2 and Sec. 4.3, respectively, for the details. Our choice of an autoregressive model stems from two main reasons: (1) they are capable of efficiently adding new nodes and edges starting from a pre-existing graph; (2) they have been shown to require less memory than one-shot models (Liao et al., 2019; Cognolato et al., 2026b), making 3DSG generation feasible. After the halting signal, the graph is returned, and the initial proximity edges are removed for the final result.

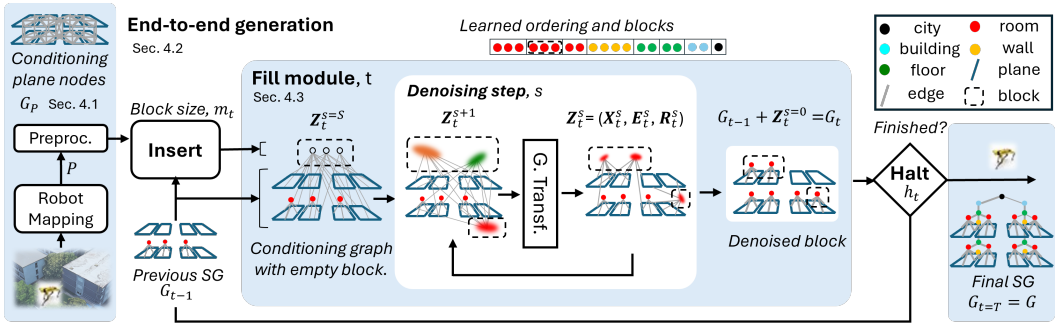

Figure 2: Plane nodes from robot mapping are preprocessed into a proximity graph $G_{\mathcal{P}}$ (left), from which the 3DSG is grown iteratively (right). At generation step t, *Insert* selects the size m_t of the next block added to G_{t-1}; *Fill* jointly denoises its classes, edges, and 3D positions from $Z_t^{s=S}$ to $Z_t^{s=0}$; and *Halt* predicts H_t, ending generation at $G_{t=T}$.

4.1 Initializing the graph of planes

As the first step of our algorithm, we augment the input set of planes with edges based on their positions, building a graph. The resulting structure provides the generative model with relational context, which is then exploited by Graph Neural Network layers (Scarselli et al., 2008; Corso et al., 2024). Formally, for each set $\mathcal{P}$, we construct a graph $G_{\mathcal{P}}$ with $\mathcal{V} = \{v_1, \dots, v_{|\mathcal{P}|}\}$, λ always returning the *plane* label, ρ returning the 3D centroids, $\pi_{\mathcal{P}}$ returning the correct plane features, and edges $\mathcal{E}_{\mathcal{P}}$ included through the following criterion:

$$\mathcal{E}_{\mathcal{P}} = \left\{ (v, v') \in \mathcal{V} \times \mathcal{V} \middle| v' \in \text{topk}_{\mathcal{V}}(-d(v, \cdot)), v \neq v' \right\}, \tag{1}$$

where $\text{topk}_{\mathcal{V}}(-d(v, \cdot))$ returns the nearest k nodes to v based on the Euclidean centroid distances.

4.2 FLAGG for graph extension

Following the preprocessing step of the previous section, we formulate the sampling of the hierarchy G from the planes as $p_\theta(G|G_{\mathcal{P}})$, i.e., the model extends the graph of planes $G_{\mathcal{P}}$, adding edges, new high-level concept nodes and their centroids. To do so, we propose the use of FLAGG, which enables the design of autoregressive models using modular components. At each step t, FLAGG adds a block of new nodes and edges to the current graph G_{t-1}, producing a larger graph G_t. This yields a sequence $G_{0:T} = (G_0, \dots, G_t, \dots, G_T)$ of increasingly large graphs, with $G_0 = G_{\mathcal{P}}$ and $G_T = G_{\text{final}}$. The total number of steps T is not fixed but determined at generation time by a binary halting signal H_t: generation continues until $H_t = 1$. Each new graph G_t at generation step t is sampled from the transition probability $p(G_t, H_t|G_{t-1})$, defined as follows: *(i)* first, the *Insertion model* $p_\phi^I(m_t|G_{t-1})$ samples the size m_t of the next *block*, i.e., the number of new nodes to be jointly added to the graph; *(ii)* the connectivity and any relevant feature, e.g., labels and centroids, of the new nodes are sampled by the *Fill model* $p_\theta^F(G_t|m_t, G_{t-1})$, producing the next graph G_t; *(iii)* the end of generation is signaled by a positive sample from the *Halting model* $p_\nu^H(H_t = 1|G_t)$. Formally, a single transition and the full distribution of FLAGG are defined as:

$$p_{\phi,\theta,\nu}(G_t, H_t = h|G_{t-1}) = p_\nu^H(H_t = h|G_t) p_\theta^F(G_t|m_t, G_{t-1}) p_\phi^I(m_t|G_{t-1}), \tag{2}$$

$$p_{\phi,\theta,\nu}(G|G_{\mathcal{P}}) = \sum_{T=1}^{\infty} \sum_{G_{0:T} \in \mathcal{I}(G, G_{\mathcal{P}}, T)} \prod_{t=1}^{T} p_{\phi,\theta,\nu}(G_t, H_t = h_t|G_{t-1}), \tag{3}$$

where $\mathcal{I}(G, G_{\mathcal{P}}, T)$ is the set of all insertion sequences of graphs $G_{0:T}$, and the halting signal h_t is 1 for $t = T$, and 0 otherwise. We implement the Insertion and Halting models as Graph Isomorphism Networks with Edges (GINE) (Hu* et al., 2020), predicting the block size m_t and the halting signal h_t, respectively. The Fill model can be any one-shot graph generative model, adapted as in Cognolato et al. (2026b). In the following section, we detail our choice of the Fill model. The FLAGG modules

are trained to recover nodes gradually removed from a data graph G through a node removal process, stopping at $G_{\mathcal{P}}$. This process characterization, comprising how many nodes are removed at each step and their order, directly influences the learned insertion policy, as it tries to reverse it. Thus, the node schedule m_t and the node ordering $\sigma : \mathcal{V} \to \{1, \dots, |\mathcal{V}|\}$ are two fundamental hyperparameters of FLAGG, which can be designed taking into consideration the prior knowledge on the problem. For the node schedule, we use the efficient categorical removals from FLAGG which, given a fixed set $\mathcal{D}$ of possible block sizes, computes the smallest multiset of block sizes that sum to the number of nodes of G minus those in $G_{\mathcal{P}}$, and samples its elements in random order. We also devise a node order that considers the hierarchy of the graph, and refer the reader to Appendix B.1.

4.3 Generating edges, labels and centroids

At each step of FLAGG, the Fill module receives as input the current partial graph G_{t-1} and the number of nodes to add m_t, and samples the edges $\mathcal{E}_t$ across new nodes and old nodes, node labels λ_t, and centroids ρ_t of the new nodes. At the end of sampling, the new features and edges are added to G_{t-1} to compose the next partial graph G_t, carrying on with the FLAGG loop. Mixed Graph and 3D Denoising Diffusion (MiDi) (Vignac et al., 2023b), a diffusion model capable of sampling both topology and 3D geometry of a graph, is our Fill module of choice. Because it works on the dense representation of a graph, made of tensors, we introduce the following objects, retrievable from the features above: the node labels $\boldsymbol{X}_t \in \{0,1\}^{m_t \times |\mathcal{C}|}$, the new-new and new-old edge tensor $\boldsymbol{E}_t \in \{0,1\}^{m_t \times (m_t + n_{t-1}) \times 2}$, and the node centroids $\boldsymbol{R}_t \in \mathbb{R}^{m_t \times 3}$. Finally, let the tensor subgraph $\boldsymbol{Z}_t = (\boldsymbol{X}_t, \boldsymbol{E}_t, \boldsymbol{R}_t)$ collect all of these tensors. $\boldsymbol{Z}_t$ only considers new nodes and connections because FLAGG stores and processes the partial graph G_{t-1} as a sparse graph, leading to lower memory usage. Using MiDi as a Fill module requires some adaptations because the generative objectives and inputs differ from those of the original work. In the following we detail how MiDi works in this adapted setup at every iteration t of FLAGG. First, the coordinate system is anchored to the centroid $\boldsymbol{c}_{t-1} = \frac{1}{n_{t-1}} \sum_{v \in \mathcal{V}_{t-1}} \rho(v)$ of G_{t-1} and its coordinates are translated as $\bar{\rho}_{t-1}(v) = \rho_{t-1}(v) - \boldsymbol{c}_{t-1}$, obtaining the centered conditioning graph $\bar{G}_{t-1}$. Second, let $s = 0, \dots, S$ denote the diffusion time step, where $\boldsymbol{Z}_t^0$ is the clean subgraph while $\boldsymbol{Z}_t^S$ is the fully noisy subgraph. Then, the MiDi module is trained to reverse the multimodal forward diffusion process at each timestep s given the partial graph G_{t-1}. At generation time, MiDi starts from a noisy sample $\boldsymbol{Z}_t^S$, and gradually denoises it until it recovers the learned output distribution, sampling $\boldsymbol{Z}_t^0$. Refer to Appendix A for more details. The neural network of the adapted MiDi is implemented by a Graph Transformer (Dwivedi & Bresson, 2020) with its layers being interleaved with GINE layers processing the nodes of G_{t-1}, which are used as conditioning information. At the end of the Fill step, after the graph G_t is obtained, its coordinates are translated back to the original frame by adding $\boldsymbol{c}_{t-1}$. FLAGG with MiDi as its Fill module is equivariant in distribution under arbitrary translations and rotations about the vertical axis. The formal statement and proof are given in Theorem 1 in Appendix I.

4.4 Additional training techniques

A potential issue carried by autoregressive architectures is exposure bias (Ranzato et al., 2015), where errors from earlier steps accumulate during generation. While the literature offers a plethora of solutions to this problem (Schmidt, 2019; Huang et al., 2026), adapting them to graphs is not trivial. To mitigate this issue, we propose two noise data augmentation schemes that teach the model to recover from imperfect choices: the first is applied to the Fill module, adding Gaussian noise to node coordinates and random perturbations in node labels and edges; the second is applied to the whole pipeline, diverging from the classical optimal categorical removal sequence used in FLAGG, and allowing suboptimal choices, mimicking wrong predictions made by the model. On the other hand, FLAGG is a modular framework, requiring a balance in gradient contributions from each model, and in the diffusion-based Fill module itself. For this reason, we reweight the sample rate of timesteps t in the removal process to favor times with higher variability of G_t, allowing the model to explore partial graphs more uniformly, and avoid overfitting to trivial cases at $t = 0$ and $t = T$. Focusing on the Fill module, we apply a loss reweighting scheme to the coordinate term in Eq. 9 based on SNR (Hang et al., 2023), increasing the contribution of almost-clean, low s timesteps and decreasing high noise contributions of high s timesteps. More details can be found in Appendix C.

5 HIERARCHICAL 3DSG GENERATION BENCHMARK

5.1 DATASETS

To the best of our knowledge, no public dataset provides the complete metric-semantic hierarchical 3DSGs targeted in this work. Therefore, we construct and publicly release our dataset[1] consisting of 9 subsets derived from three sources (Tab. 1): (i) a synthetic generator (S), which allows controlled layout complexity and for which we define graph structure and node centroids by construction; (ii) the publicly available MSD dataset (M) (van Engelenburg et al., 2024), which provides structure and centroids for walls and rooms, while higher-level nodes (floor and above) are constructed; and (iii) real environments (R) recorded by a robot with a LiDAR sensor, where only planes are observed, and thus the full hierarchy (walls, rooms, and higher levels) is manually constructed. Whenever structure and/or centroids are missing in the source, we infer them using the same node labeling and centroid definition functions, $\lambda(\cdot)$ and $\rho(\cdot)$, ensuring a consistent construction procedure across all sources. For each source, we instantiate subsets at increasing hierarchy depth: up to the floor level (-F), the building level (-B), or the city level (-C). The S-F-□ variant further restricts synthetic layouts to rectangular rooms only. Layout complexity and graph size grow with hierarchy depth across all sources, while the average node degree remains approximately constant due to the shared tree-like structure. The MSD and real sources exhibit substantially larger and more complex graphs than the synthetic source. The real source is small and used only for testing.

5.2 METRICS

Generated 3DSGs are compared with ground truth, striving for similar topological statistics and correctness. Thus, we define a set of *graph matching metrics*, comparing the sampled extension of a set of planes with its ground truth, and a set of *distribution metrics*, computing the similarity in distribution of topological properties across the two sets of data.

Graph matching evaluation. The output of the model must align with the ground truth hierarchy and centroids, which requires mapping each generated node to a ground truth node even when the number of nodes or the structures differ. We obtain this mapping through the Fused Gromov-Wasserstein (FGW) distance (Vayer et al., 2020), which finds the transport plan between generated and ground truth nodes minimizing a cost that combines the Wasserstein distance (W), matching nodes by their features, and the Gromov-Wasserstein distance (GW), accounting for the structure:

$$d_{FGW,\alpha}(G, G') = \inf_{\gamma \in \Gamma} (1-\alpha) d_W(\gamma, G, G') + \alpha d_{GW}(\gamma, G, G'), \tag{4}$$

where $\gamma : \mathcal{V} \times \mathcal{V}' \to [0, 1]$ is a transport plan in the set of all plans Γ, and $\alpha = 0.5$. The W distance matches nodes by their centroids and one-hot encoded labels, while the GW distance matches pairs of nodes by their shortest-path lengths SP in G and G', as a measurement of graph topology similarity:

$$d_W(\gamma, G, G') = \sum_{(v,v') \in \mathcal{V} \times \mathcal{V}'} \gamma(v, v') \left\| \begin{pmatrix} \rho(v) \\ \text{onehot}(\lambda(v)) \end{pmatrix} - \begin{pmatrix} \rho'(v') \\ \text{onehot}(\lambda'(v')) \end{pmatrix} \right\|, \tag{5}$$

$$d_{GW}(\gamma, G, G') = \sum_{\substack{(v,w) \in \mathcal{V} \times \mathcal{V} \\ (v',w') \in \mathcal{V}' \times \mathcal{V}'}} \gamma(v, v') \gamma(w, w') \| \text{SP}(v, w) - \text{SP}(v', w') \|. \tag{6}$$

To allow a 1-to-1 matching between the two graphs, we pad the smallest with $|n - n'|$ isolated dummy nodes whose pairwise SP lengths to all other nodes are set to one plus the maximum SP in the original graph. This ensures that dummy nodes are penalized by both the W and GW terms, so the transport plan avoids matching real nodes to dummy nodes unless a better alignment exists.

Distribution evaluation. Evaluating distribution similarity is non-trivial in graphs, as they present a broad range of properties (Thompson et al., 2022). Current metrics are based on the Maximum Mean Discrepancy (MMD) comparing the distance between the distributions of a feature. Please find details in Appendix H. As features we consider: the node degree histogram (Deg.), the clustering coefficient histogram (Clust.), the spectrum (Spec.), and the embeddings obtained by a randomly initialized Graph Isomorphism Network (GIN) (Xu et al., 2019).

[1] https://anonymous.4open.science/r/3DSG-generation-datasets-2787

Table 1: **Datasets.** Each level includes all the levels below it: floor-level graphs contain walls, rooms and floors, building-level graphs add buildings, and city-level graphs add cities. Per-graph averages.

Source	**Synthetic**			**MSD**			**Real**		
Highest level	Floor	Floor	City	Floor	Building	City	Floor	Building	City
Dataset	S-F-□ (□ rooms)	S-F	S-C	M-F	M-B	M-C	R-F	R-B	R-C
# graphs	2,400	2,999	2,999	3,466	2,416	1,084	4	1	1
$\lvert\mathcal{V}\rvert$	23	30	110	95	159	223	45	135	172
$\lvert\mathcal{E}\rvert$	26.2	34.8	130.5	112.3	186.8	258.6	46.0	145.0	185.0
Degree	2.28	2.34	2.38	2.36	2.35	2.32	2.04	2.13	2.11

6 EXPERIMENTS

We evaluate all models and baselines on the proposed 3DSG benchmark (see Section 5). Specifically, we split each dataset in Section 5.1 into 60% for training, 20% for validation and 20% for test. We train our models on the training set, and select the checkpoint scoring the best FGW on the validation set. Finally, we report the metrics in Section 5.2 computed on the test set in Table 2 and Table 3. All hyperparameters can be found in Appendix B.2. We run all experiments with three seeds and report the mean and standard deviation across them. Experiments were run on NVIDIA V100-32GB GPUs. We make our code publicly available[2].

6.1 BASELINES

We compare with two learning-based methods for this task, EC-□ and EC-L (Sec. 2), and a one-shot diffusion model; Tab. 6 summarizes their capabilities. We exclude geometry-driven pipelines that operate on raw sensor maps rather than on graphs of planes. EC-□ (rectangular rooms) and EC-L (L-shaped rooms) are trained on their original data (S-F-□ and S-F, respectively) and, since they generate no floors, are evaluated on S-F-□, S-F, M-F and R-F with floor nodes removed, over three random seeds. Finally, to compare against the state of the art in graph generation, we include MIDI+OR, the one-shot model MiDi (Vignac et al., 2023b) adapted to conditional 3DSG completion as described in Sec. E.1. Since one-shot models require the final graph size as input, MIDI+OR receives the correct number of nodes to be generated from an oracle (Or), an advantage not available to autoregressive methods. MIDI+OR generates the full hierarchy and is therefore evaluated on all datasets. We regard MIDI+OR as a reference under privileged information rather than a directly deployable method. Since MIDI+OR relies on the same denoising model, the comparison between the two highlights the effect of the generation scheme, autoregressive versus one-shot. We provide an additional random baseline as upper-bound reference in Appendix. E.2

6.2 RESULTS

Tab. 2 and Fig. 3 report the results on the synthetic and MSD sources. OURS attains the lowest FGW on every dataset but M-F. Against EC-□ and EC-L, the gain is large even on the single-floor datasets on which they were trained (0.8–1.1 vs. 8.0–13.0 in FGW) and grows with layout complexity: on M-F, FGW, GW and W drop by 78–80% relative to EC-L, i.e., structure and metric-semantic attributes improve alike. The distribution metrics agree: on M-F, Degree and Spec. drop by 93% and 98%, and GIN from 6.0 to 2.3. On clustering, OURS matches the zero error that EC-□ and EC-L obtain by construction, while learning the full adjacency. More importantly, OURS surpasses MIDI+OR, which shares the same denoising model but receives the target graph size from an oracle, on every dataset but M-F, and the gap widens with depth (S-C: 3.0 vs. 6.0; M-B: 4.8 vs. 9.3; M-C: 7.3 vs. 13.3). The advantage of MIDI+OR lies mainly in GW on single-floor datasets, where it scores close to zero, as expected when the exact number of nodes is given, and yields a lower FGW on M-F (3.5 vs. 4.3). Beyond one floor, however, its errors grow and become unstable across seeds (M-C: FGW 13.3 ± 13.6, Clust. 72.2 ± 42.2), whereas OURS degrades gracefully (FGW 4.3, 4.8, 7.3 on M-F, M-B, M-C) with a standard deviation never above 0.7. Further results on memory and generation time can be found in Appendix G.

[2] https://anonymous.4open.science/r/3dsg-gen-iclr2027

Table 2: **Evaluation results for synthetic and MSD datasets.** Values are multiplied by 10^2 and reported as mean $\pm$ standard deviation over three seeds. Missing values (-) occur when the dataset contains hierarchy levels out of the scope of the baseline. Best results are highlighted in light gray and bold; second-best results in lighter gray.

Source: Synthetic

Method	S-F-□							S-F							S-C						
	FGW ↓	GW↓	W↓	Deg.↓	Spec.↓	Clust.↓	GIN↓	FGW ↓	GW↓	W↓	Deg.↓	Spec.↓	Clust.↓	GIN↓	FGW ↓	GW↓	W↓	Deg.↓	Spec.↓	Clust.↓	GIN↓
EC-□	12.0 ± 1.0	17.0 ± 1.0	8.0 ± 0.0	18.0 ± 1.0	12.0 ± 1.0	**0.0 ± 0.0**	40.0 ± 10.0	13.0 ± 1.0	17.0 ± 1.0	8.0 ± 1.0	19.0 ± 2.0	12.0 ± 1.0	**0.0 ± 0.0**	36.0 ± 9.0	-	-	-	-	-	-	-
EC-L	8.0 ± 0.0	7.0 ± 0.0	9.0 ± 0.0	25.0 ± 2.0	8.0 ± 1.0	**0.0 ± 0.0**	16.0 ± 1.0	8.0 ± 0.0	7.0 ± 0.0	9.0 ± 0.0	26.0 ± 2.0	8.0 ± 1.0	**0.0 ± 0.0**	20.0 ± 2.0	-	-	-	-	-	-	-
MiDi+Or	3.4 ± 0.1	**0.1 ± 0.2**	6.7 ± 0.1	**0.2 ± 0.1**	**0.5 ± 0.1**	**0.0 ± 0.0**	**0.2 ± 0.0**	3.3 ± 0.0	**0.0 ± 0.0**	6.5 ± 0.0	**0.4 ± 0.1**	**0.5 ± 0.1**	**0.0 ± 0.0**	**0.4 ± 0.0**	6.0 ± 0.3	7.3 ± 1.3	4.8 ± 0.8	33.3 ± 25.8	6.8 ± 1.8	1.4 ± 1.0	12.7 ± 10.5
Ours	**0.8 ± 0.1**	0.7 ± 0.1	**1.0 ± 0.1**	0.3 ± 0.1	1.0 ± 0.1	**0.0 ± 0.0**	0.3 ± 0.0	**1.1 ± 0.1**	1.0 ± 0.1	**1.3 ± 0.1**	**0.4 ± 0.0**	0.6 ± 0.0	**0.0 ± 0.0**	**0.4 ± 0.0**	**3.0 ± 0.5**	**3.2 ± 0.8**	**2.8 ± 0.3**	**2.8 ± 0.8**	**1.6 ± 0.1**	**0.1 ± 0.2**	**2.0 ± 1.1**

Source: MSD

Method	M-F							M-B							M-C						
	FGW ↓	GW↓	W↓	Deg.↓	Spec.↓	Clust.↓	GIN↓	FGW ↓	GW↓	W↓	Deg.↓	Spec.↓	Clust.↓	GIN↓	FGW ↓	GW↓	W↓	Deg.↓	Spec.↓	Clust.↓	GIN↓
EC-□	27.0 ± 0.0	27.0 ± 0.0	28.0 ± 0.0	80.0 ± 0.0	116.0 ± 0.0	**0.0 ± 0.0**	100.0 ± 1.0	-	-	-	-	-	-	-	-	-	-	-	-	-	-
EC-L	20.0 ± 0.0	18.0 ± 0.0	23.0 ± 0.0	31.0 ± 0.0	52.0 ± 0.0	**0.0 ± 0.0**	6.0 ± 0.0	-	-	-	-	-	-	-	-	-	-	-	-	-	-
MiDi+Or	**3.5 ± 0.1**	**0.2 ± 0.0**	6.9 ± 0.2	3.3 ± 1.7	1.4 ± 0.2	**0.0 ± 0.0**	**1.2 ± 0.3**	9.3 ± 8.1	9.7 ± 14.0	8.9 ± 2.2	46.8 ± 42.8	15.0 ± 15.0	28.8 ± 42.5	15.8 ± 16.6	13.3 ± 13.6	20.7 ± 26.9	**6.0 ± 0.5**	59.8 ± 17.7	36.1 ± 40.9	72.2 ± 42.2	41.7 ± 24.7
Ours	4.3 ± 0.0	3.9 ± 0.1	**4.6 ± 0.1**	**2.2 ± 0.4**	**1.0 ± 0.1**	**0.0 ± 0.0**	2.3 ± 0.5	**4.8 ± 0.1**	**4.5 ± 0.1**	**5.1 ± 0.1**	**5.4 ± 1.7**	**2.9 ± 0.5**	**0.0 ± 0.0**	**4.2 ± 2.4**	**7.3 ± 0.7**	**7.0 ± 0.4**	7.6 ± 1.1	**11.4 ± 3.1**	**8.4 ± 3.3**	**0.2 ± 0.2**	**21.0 ± 19.0**

Table 3: **Results in real environments.**

Source: Real [$\times 10^{-2}$]

Method	R-F			R-B			R-C		
	FGW ↓	GW↓	W↓	FGW ↓	GW↓	W↓	FGW ↓	GW↓	W↓
EC-□	30.4 ± 0.0	38.0 ± 0.0	22.9 ± 0.0	-	-	-	-	-	-
EC-L	22.0 ± 0.0	27.0 ± 0.0	17.0 ± 0.0	-	-	-	-	-	-
MiDi+Or	33.7 ± 3.7	53.3 ± 10.1	**14.1 ± 2.9**	25.4 ± 21.8	36.8 ± 40.5	14.0 ± 6.4	38.1 ± 9.8	61.7 ± 22.4	14.5 ± 3.3
Ours	**13.2 ± 1.3**	**9.9 ± 0.9**	16.5 ± 1.8	**8.5 ± 3.6**	**6.5 ± 4.5**	**10.5 ± 3.4**	**10.1 ± 5.4**	**10.8 ± 6.9**	**9.4 ± 4.4**

Table 4: **Ablation study in S-C**. At 40k training steps.

Method	S-C [$\times 10^{-2}$]						
	FGW ↓	GW↓	W↓	Deg.↓	Spec.↓	Clust.↓	GIN↓
w/o noisy blocks	6.89	7.68	6.09	25.52	5.71	**0.00**	4.35
w/o noise aug.	12.28	11.28	13.28	44.33	16.11	1.24	3.75
Ours-FD4 filler	**5.78**	**5.76**	5.80	27.84	7.18	0.41	11.64
Ours	6.10	6.86	**5.34**	**18.73**	**4.89**	0.05	**3.11**

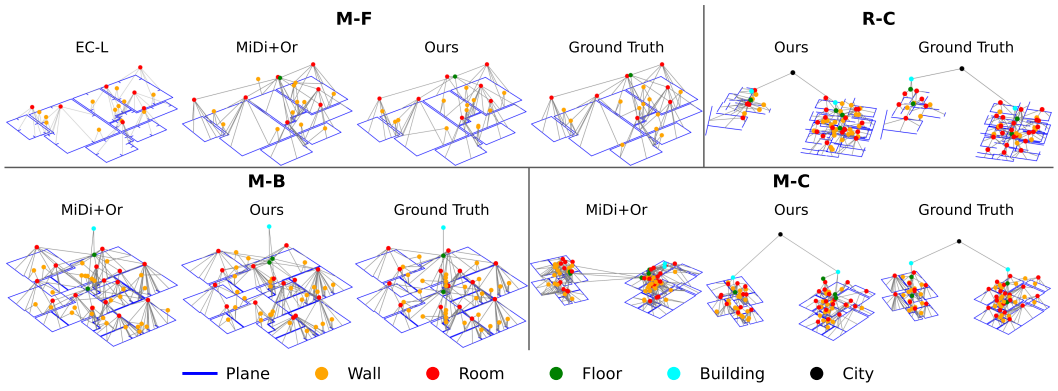

Figure 3: **Qualitative results.** Selection of the most complex datasets at the three hierarchy levels explored, comparing our model with the best-performing baselines. Full comparison in the Appendix F.

On robot-recorded environments (Tab. 3), evaluated with the models trained on the MSD subset of matching depth, all learned methods lose accuracy, but to very different extents. Ours roughly triples its floor-level FGW (4.3 on M-F to 13.2 on R-F), while R-B and R-C degrade less (8.5 and 10.1). MiDi+Or loses most: on R-F its FGW rises from 3.5 to 33.7, falling behind EC-L, with the structural term reaching GW 53.3, and on R-B it becomes unstable across seeds (25.4 ± 21.8). Ours thus attains the lowest FGW on every subset, 40% below EC-L on R-F and 67–74% below MiDi+Or on R-B and R-C, and the oracle keeps an advantage only in W on R-F. Fig. 3 and Appendix F show the failure mode: MiDi+Or attaches rooms and floors to planes across the whole scene, whereas Ours preserves the local grouping of the ground truth.

6.3 ABLATIONS

We ablate the two exposure bias mitigation strategies of Sec. 4.4 on the validation set of S-C, which combines non-rectangular layouts with full hierarchy depth. The variants in this study are trained for 40k steps, differently from other experiments which are trained to convergence. Two leave-one-out variants remove either the perturbation of the removal sequence (*w/o noisy blocks*) or that of the conditioning graph (*w/o noise aug.*). A third variant keeps both perturbations but replaces MiDi with FD4 in the Fill module (*Ours-FD4 filler*), isolating how block geometry is represented: instead of denoising the 3D coordinates of new nodes, FD4 denoises edge distances and recovers centroids via Multidimensional Scaling (Appendix D).

As Tab. 4 shows, both noise schemes improve sampling, consistent with exposure bias: without noise augmentation FGW doubles (12.28 vs. 6.10) and every metric worsens, while without noisy blocks FGW rises to 6.89, mainly through degree and GIN. OURS-FD4 FILLER reaches the lowest FGW (5.78) through a lower GW, but a higher W and clearly worse distribution metrics (GIN 11.64 vs. 3.11), suggesting that recovering centroids from denoised distances is less accurate and that its per-graph structural matches do not yield realistic graph statistics.

7 DISCUSSION AND LIMITATIONS

Three conclusions follow from Sec. 6. First, jointly learning structure, labels and centroids in one model outperforms the edge-classification pipelines, most on the layouts that break their assumptions (concepts composed of a variable number of lower-level nodes), and removes their ceiling: EC-□ and EC-L cannot represent anything above rooms, and their zero clustering error only reflects that they emit two edge types, whereas OURS learns the full adjacency and still attains it. Second, the autoregressive scheme improves generation itself, beyond enabling variable graph sizes: with the same denoising model and the ground-truth number of nodes, one-shot generation is competitive only on single floors and falls behind, with growing variance, as the hierarchy deepens. Since every concept in a 3DSG is defined by the lower-level nodes it aggregates, generating one block at a time conditions each new node on the layer that supports it, whereas one-shot generation must resolve all layers and their dependencies at once. Third, the remaining error is driven mainly by layout complexity: FGW quadruples from synthetic to MSD single-floor scenes but rises only mildly when floors, buildings and cities are added, so the same architecture, trained per hierarchy level without concept-specific modules, extends to deeper hierarchies at a moderate cost. The largest source of error is the shift from floor plans to sensor data, which affects every method; that the one-shot model loses most of its accuracy there while OURS retains most of it suggests that conditioning on the layer below makes generation less sensitive to the input geometry. Finally, the proposed FGW-based metrics proved useful for assessing topological and spatial similarity with respect to ground truth hierarchies. They are complementary to standard graph generation indicators, offering 3DSG-specific measures of quality: GW for the structure, W for metric-semantic fidelity, FGW as a well-balanced single indicator for 3DSG generation.

8 CONCLUSION

We propose the first autoregressive diffusion-based DGG model for 3DSG completion, which grows a hierarchy of walls, rooms, floors, buildings and cities from robot-observed planes, jointly generating structure, labels and centroids without concept-specific modules. Across synthetic scenes, architectural floor plans, and robotic sensor data, it outperforms existing learning-based baselines on single-floor generation, extends to multi-floor, multi-building and city scenarios where no prior learning-based method applies, and, without access to the target graph size, surpasses a one-shot model that receives it from an oracle on all but one dataset, with the gap widening as the hierarchy deepens. We further advocate Fused Gromov–Wasserstein (FGW) as a unified metric for 3DSG generation and publicly release our dataset and code. Two limitations point to future work. The shift from floor plans to robot-recorded planes remains the largest source of error for all methods, OURS included; since real data are used only for testing, training on or adapting to sensor-derived planes is a natural next step. Moreover, our setting is offline and bottom-up, so an early error propagates upward through the hierarchy; online generation that refines the hierarchy as new planes arrive would address both and enable use during mapping.

AI USE STATEMENT

In this work, we used generative AI tools to aid or polish writing, research ideation or execution, and for retrieval and discovery (e.g., finding related work). We have not used generative AI tools for drafting sections of the paper, generating synthetic datasets, or proving mathematical claims. Additionally, we used generative AI tools to revise our own code for errors. We have reviewed all AI-assisted work. We checked LLM-generated research ideas for potential plagiarism through a manual literature survey and LLM-generated code was verified and tested for correctness by the two first authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have taken several steps to make our results reproducible. The complete source code, including training, sampling, and evaluation scripts, is available at an anonymous repository (Sec. 6), and the hierarchical 3DSG datasets used in all experiments, covering the synthetic, MSD-derived, and robot-recorded subsets, are publicly released (Sec. 5). Sec. 5.1 and Tab. 1 describe how each subset is constructed, including how missing structure and centroids are completed with a consistent labeling and centroid definition across sources. The model is specified in Section 4, with the node orderings in Appendix B.1, the training techniques in Appendix C, the FD4 variant in Appendix D, and the baselines in Appendix E.1. Appendix B.2 includes a comprehensive list of relevant hyperparameter of our model and of MiDi+Or, together with the configuration file in the released code that sets each value, so that each run can be reproduced from a single preset. Section 6 specifies the data splits, the checkpoint-selection criterion, the number of seeds, and the hardware. Appendix G reports sampling time and memory, and the metrics are defined in Sec. 5.2.

REFERENCES

Iro Armeni, Zhi-Yang He, JunYoung Gwak, Amir R Zamir, Martin Fischer, Jitendra Malik, and Silvio Savarese. 3D Scene Graph: A structure for unified semantics, 3D space, and camera. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5664–5673, 2019.

Hriday Bavle, Jose Luis Sanchez-Lopez, Muhammad Shaheer, Javier Civera, and Holger Voos. S-graphs+: Real-time localization and mapping leveraging hierarchical representations. *arXiv preprint arXiv:2212.11770*, 2022.

Hriday Bavle, José Luis Sánchez-López, Muhammad Shaheer, Javier Civera, and Holger Voos. S-graphs 2.0 – a hierarchical-semantic optimization and loop closure for slam. *IEEE Robotics and Automation Letters*, 10:12461–12468, 2025. URL https://api.semanticscholar.org/CorpusID:276580192.

Samuel Cognolato, Davide Rigoni, Marco Ballarini, Luciano Serafini, Stefano Moro, and Alessandro Sperduti. D4: Distance diffusion for a truly equivariant molecular design. *Neurocomputing*, 679:133197, 2026a. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2026.133197. URL https://www.sciencedirect.com/science/article/pii/S0925231226005941.

Samuel Cognolato, Alessandro Sperduti, and Luciano Serafini. Flagg: Flexible autoregressive graph generation. *Journal of Machine Learning Research*, 27(107):1–38, 2026b. URL http://jmlr.org/papers/v27/25-0994.html.

Gabriele Corso, Hannes Stark, Stefanie Jegelka, Tommi Jaakkola, and Regina Barzilay. Graph neural networks. *Nature Reviews Methods Primers*, 4(1):17, 2024.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

Saad Ejaz, Marco Giberna, Muhammad Shaheer, Jose Andres Millan-Romera, Ali Tourani, Paul Kremer, Holger Voos, and Jose Luis Sanchez-Lopez. Situationally-aware path planning exploiting 3d scene graphs. *arXiv preprint arXiv:2508.06283*, 2025.

Miguel Fernandez-Cortizas, Hriday Bavle, David Perez-Saura, Jose Luis Sanchez-Lopez, Pascual Campoy, and Holger Voos. Multi s-graphs: An efficient distributed semantic-relational collaborative slam. *IEEE Robotics and Automation Letters*, 9(6):6004–6011, 2024.

Aditya Grover, Aaron Zweig, and Stefano Ermon. Graphite: Iterative generative modeling of graphs. In *International conference on machine learning*, pp. 2434–2444. PMLR, 2019.

Tiankai Hang, Shuyang Gu, Chen Li, Jianmin Bao, Dong Chen, Han Hu, Xin Geng, and Baining Guo. Efficient diffusion training via min-snr weighting strategy. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 7407–7417. IEEE, 2023.

Weihua Hu*, Bowen Liu*, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJlWWJSFDH.

Xun Huang, Zhengqi Li, Guande He, Mingyuan Zhou, and Eli Shechtman. Self forcing: Bridging the train-test gap in autoregressive video diffusion. *Advances in Neural Information Processing Systems*, 38:167283–167308, 2026.

Nathan Hughes, Yun Chang, and Luca Carlone. Hydra: A real-time spatial perception system for 3d scene graph construction and optimization. In *Robotics: Science and Systems*, 2022. doi: 10.15607/RSS.2022.XVIII.050.

Lingkai Kong, Jiaming Cui, Haotian Sun, Yuchen Zhuang, B Aditya Prakash, and Chao Zhang. Autoregressive diffusion model for graph generation. In *International conference on machine learning*, pp. 17391–17408. PMLR, 2023.

Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. *Advances in neural information processing systems*, 32, 2019.

Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11451–11461, 2022. doi: 10.1109/CVPR52688.2022.01117.

Matteo Luperto, Tomasz Piotr Kucner, Andrea Tassi, Martin Magnusson, and Francesco Amigoni. Robust structure identification and room segmentation of cluttered indoor environments from occupancy grid maps. *IEEE Robotics and Automation Letters*, 7(3):7974–7981, 2022.

Łukasz Maziarka, Agnieszka Pocha, Jan Kaczmarczyk, Krzysztof Rataj, Tomasz Danel, and Michał Warchoł. Mol-cyclegan: a generative model for molecular optimization. *Journal of Cheminformatics*, 12(1):2, 2020.

Jose Andres Millan-Romera, Hriday Bavle, Muhammad Shaheer, Martin R Oswald, Holger Voos, and Jose Luis Sanchez-Lopez. Learning high-level semantic-relational concepts for slam. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9803–9810. IEEE, 2024.

Jose Andres Millan-Romera, Muhammad Shaheer, Miguel Fernandez-Cortizas, Martin R Oswald, Holger Voos, and Jose Luis Sanchez-Lopez. Generation of uncertainty-aware high-level spatial concepts in factorized 3d scene graphs via graph neural networks. *IEEE Robotics and Automation Letters*, 2026.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Florian Schmidt. Generalization in generation: A closer look at exposure bias. In Alexandra Birch, Andrew Finch, Hiroaki Hayashi, Ioannis Konstas, Thang Luong, Graham Neubig, Yusuke Oda, and Katsuhito Sudoh (eds.), *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pp. 157–167, Hong Kong, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-5616. URL https://aclanthology.org/D19-5616/.

Muhammad Shaheer, Jose Andres Millan-Romera, Hriday Bavle, Marco Giberna, Jose Luis Sanchez-Lopez, Javier Civera, and Holger Voos. Tightly coupled slam with imprecise architectural plans. *IEEE Robotics and Automation Letters*, 10(8):8019–8026, 2025. doi: 10.1109/LRA.2025.3582108.

Rajat Talak, Siyi Hu, Lisa Peng, and Luca Carlone. Neural trees for learning on graphs. *Advances in Neural Information Processing Systems*, 34:26395–26408, 2021.

Rylee Thompson, Boris Knyazev, Elahe Ghalebi, Jungtaek Kim, and Graham W Taylor. On evaluation metrics for graph generative models. In *International Conference on Learning Representations*, 2022.

Warren S Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17(4):401–419, 1952.

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, 2024.

Titouan Vayer, Laetitia Chapel, Rémi Flamary, Romain Tavenard, and Nicolas Courty. Fused gromov-wasserstein distance for structured objects. *Algorithms*, 13(9):212, 2020.

Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. Digress: Discrete denoising diffusion for graph generation. In *International Conference on Learning Representations*, 2023a.

Clement Vignac, Nagham Osman, Laura Toni, and Pascal Frossard. Midi: Mixed graph and 3d denoising diffusion for molecule generation. In *ECML PKDD*, pp. 560–576. Springer, 2023b.

Shun-Cheng Wu, Johanna Wald, Keisuke Tateno, Nassir Navab, and Federico Tombari. Scenegraphfusion: Incremental 3d scene graph prediction from rgb-d sequences. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7515–7525, 2021.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=ryGs6iA5Km`.

Michael Ying Yang, Wentong Liao, Hanno Ackermann, and Bodo Rosenhahn. On support relations and semantic scene graphs. *ISPRS journal of photogrammetry and remote sensing*, 131:15–25, 2017.

Chengxi Zang and Fei Wang. Moflow: an invertible flow model for generating molecular graphs. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 617–626, 2020.

A MiDi as Fill Module

In this section we present the technical details of the MiDi training and generation. The forward diffusion transition is:

$$q(\boldsymbol{Z}_t^s|\boldsymbol{Z}_t^{s-1}) = \mathrm{Cat}(\boldsymbol{X}_t^{s-1}\boldsymbol{Q}_X^s) \times \mathrm{Cat}(\boldsymbol{E}_t^{s-1}\boldsymbol{Q}_E^s) \times \mathcal{N}(\alpha^s \boldsymbol{R}_t^{s-1}, (\sigma^s)^2 \boldsymbol{I}), \tag{7}$$

where $\boldsymbol{Q}_X^s$ and $\boldsymbol{Q}_E^s$ are the transition matrices for the discrete diffusion of node labels and edges, respectively, while α^s and σ^s are the scaling and noise parameters for the Gaussian diffusion of coordinates. The transitions are built to make the process converge to an easy-to-sample noisy subgraph $\boldsymbol{Z}_t^S$. The role of the generative model is to reverse this process starting from $\boldsymbol{Z}_t^S$ and the conditioning graph G_{t-1}, producing $\boldsymbol{Z}_t^0$. The reverse transitions are:

$$\begin{aligned}
p_\theta(\boldsymbol{X}_t^{s-1}|\boldsymbol{Z}_t^s, \bar{G}_{t-1}) &= \sum\nolimits_{\boldsymbol{X}_t^0} q(\boldsymbol{X}_t^{s-1}|\boldsymbol{X}_t^s, \boldsymbol{X}_t^0) p_\theta^X(\boldsymbol{X}_t^0|\boldsymbol{Z}_t^s, \bar{G}_{t-1}); \\
p_\theta(\boldsymbol{E}_t^{s-1}|\boldsymbol{Z}_t^s, \bar{G}_{t-1}) &= \sum\nolimits_{\boldsymbol{E}_t^0} q(\boldsymbol{E}_t^{s-1}|\boldsymbol{E}_t^s, \boldsymbol{E}_t^0) p_\theta^E(\boldsymbol{E}_t^0|\boldsymbol{Z}_t^s, \bar{G}_{t-1}); \\
p_\theta(\boldsymbol{R}_t^{s-1}|\boldsymbol{Z}_t^s, \bar{G}_{t-1}) &= \mathcal{N}(\mu^s \hat{\boldsymbol{R}}_\theta(\boldsymbol{Z}_t^s, \bar{G}_{t-1}) + \nu^s \boldsymbol{R}_t^s,\ (\tilde{\sigma}^s)^2 \boldsymbol{I});
\end{aligned} \tag{8}$$

where $p_\theta(\boldsymbol{X}_t^0|\boldsymbol{Z}_t^s, \bar{G}_{t-1})$, $p_\theta(\boldsymbol{E}_t^0|\boldsymbol{Z}_t^s, \bar{G}_{t-1})$, and $\hat{\boldsymbol{R}}_\theta(\boldsymbol{Z}_t^s, \bar{G}_{t-1})$ are the predicted clean node labels, edges, and coordinates, respectively, while μ^s, ν^s, and $\tilde{\sigma}^s$ are the Gaussian transition parameters. The model, parameterized by θ, is trained to maximize the log-likelihood of the reverse transitions, which is equivalent to minimizing the following loss:

$$\mathcal{L}(\theta) = \lambda_x \,\mathrm{CE}(\boldsymbol{X}_t^0, p_\theta^X(\cdot|\boldsymbol{Z}_t^s, \bar{G}_{t-1})) + \lambda_e \,\mathrm{CE}(\boldsymbol{E}_t^0, p_\theta^E(\cdot|\boldsymbol{Z}_t^s, \bar{G}_{t-1})) + \lambda_r \|\boldsymbol{R}_t^0 - \hat{\boldsymbol{R}}_\theta(\boldsymbol{Z}_t^s, \bar{G}_{t-1})\|_2^2, \tag{9}$$

where λ_x, λ_e, and λ_r are the weights for the node labels, edges, and coordinates losses, respectively, and CE is the cross-entropy loss.

B Architectural design

B.1 Orderings for 3DSGs

The ordering $\sigma : \mathcal{V} \to \{1, \ldots, |\mathcal{V}|\}$ in FLAGG is used to define the removal order, which in turn is learned backward. To facilitate understanding, we will refer to σ as the intended order of *insertion*, instead of removal. As illustrated in Eq. 3, the probability of a graph G is influenced by the different orderings in which it is generated (Liao et al., 2019). As stated in Section 3, it is known that the nodes of each layer in the hierarchy directly depend on their children in the layer below. For this reason, a sensible way to order nodes is to do so by layer in a bottom-up fashion. We refer to it as Hierarchical ordering (H). First, an order on the concepts $\sigma_\mathcal{C} : \mathcal{C} \to \{1, \ldots, |\mathcal{C}|\}$ must be defined. Then, the ordered layers are defined as

$$\mathcal{L}_i = \{v \in \mathcal{V} : \sigma_\mathcal{C}(\lambda(v)) = i\}, \quad \forall i = 1, \ldots, |\mathcal{C}| \tag{10}$$

Let $\sigma_U(\mathcal{L}) : \mathcal{L} \to \{1, \ldots, |\mathcal{L}|\}$ be a uniformly random ordering of all the nodes in a generic $\mathcal{L}$. The Hierarchical ordering is given by:

$$\sigma_\mathrm{H}(v) = \sum_{i=1}^{k-1} |\mathcal{L}_i| + \sigma_U(\mathcal{L}_k)(v) \quad \text{with } v \in \mathcal{L}_k. \tag{11}$$

We additionally propose a Hierarchical Proximity ordering (HP), which replaces $\sigma_U(\mathcal{L})$ with an ordering by proximity $\sigma_P(\mathcal{L})$, starting from a random node $v_\text{start} \sim \mathcal{U}(\mathcal{L})$:

$$\sigma_P(\mathcal{L})(v) = \Big|\big\{w \in \mathcal{L} : d(v_\text{start}, w) \le d(v_\text{start}, v)\big\}\Big|. \tag{12}$$

In the case where some nodes are assigned the same index, i.e., when they have exactly the same distance from the starting node, a random order among them is sampled. The aim of the HP ordering is to guide generation from the first node of a layer outwards.

B.2 Hyperparameters

We provide a summary of the main hyperparameters of the proposed model in Table 5. The full list of hyperparameters can be found as Hydra configuration files in the code repository.

Table 5: Hyperparameters of Ours (FLAGG with a MiDi-based Fill module) and of the MiDi+Or baseline, identical for all datasets. Presets: `preset/final/midi2/base.yaml` (Ours) and `preset/final/midi/base.yaml` (MiDi+Or), relative to `config/`.

Hyperparameter	Value
Ours — Insertion and Halting models	
Architecture	GINE, 5 layers, 128 hidden
Ours — node removal process	
Node ordering σ	scene-graph proximity permutation
Node schedule	categorical removals, class order $(0, 4, 5, 3, 2, 1)$
Block sizes $\mathcal{D}$	$\{1, 4\}$
Noisy blocks	block-size noise $p = 0.3$ (uniform)
Weighted subsampling	$\sqrt{\cdot}$-weighted sampling of t, subsampling by weight mass, keep t=1
Ours — Fill module (MiDi-based denoiser)	
Architecture	graph transformer with positions, 9 layers, 8 heads
Conditioning GINE	1 layer, 256 hidden, for each graph transformer layer
Diffusion	$S = 500$, cosine-ν schedule, $\nu_X = 1$, $\nu_E = 1.5$, $\nu_R = 2.5$
Noise processes	marginal discrete ($\mathbf{X}$, $\mathbf{E}$), VP Gaussian ($\mathbf{R}$)
Loss weights (Eq. 9)	$\lambda_x = 1$, $\lambda_e = 2$, $\lambda_r = 4$
Min-SNR	$\gamma = 5$ on the coordinate term
Ours — training and sampling	
Noise augmentation	$p_{\max} = 0.5$ (1k-step warm-up); node flip 0.01, edge drop 0.05, edge flip 0.025, edge add 0.01, position noise $\sigma = 0.1$
Model selection	best validation FGW, checkpoint every 250 steps
Sampling batch	200 graphs
MiDi+Or	
Architecture	graph transformer, 9 layers, 8 heads
Diffusion	$S = 500$, cosine schedule, marginal transitions, $\nu_X = 1$, $\nu_E = 1.5$, $\nu_R = 2.5$
Loss weights (Eq. 9)	$\lambda_x = 0.4$, $\lambda_e = 1$, $\lambda_r = 5$
Model selection	as Ours
Evaluation (both methods; FGW also drives model selection)	
FGW trade-off	$\alpha = 0.5$ (structure vs. features)

C Training techniques

C.1 Mitigating exposure bias

Noise augmentation. During training, FLAGG is trained in a teacher-forcing regime, i.e., receiving ground-truth conditioning graphs, while at generation time G_{t-1} contains predictions from earlier steps, which might differ from the training data. We do observe exposure bias in our preliminary tests, leading to worse generation quality in floor and building datasets. Thus, we perturb the training-time conditioning graphs to simulate this effect. The observed plane nodes are protected; previously generated high-level nodes are not. For each non-protected node v, we replace its label with another label in $\mathcal{C} \setminus \{\textit{plane}, \lambda(v)\}$ with probability p_{node}, and add Gaussian position noise with scale σ_{pos}:

$$\widetilde{\rho}(v) = \rho(v) + \boldsymbol{\epsilon}_v, \qquad \boldsymbol{\epsilon}_v \sim \mathcal{N}(\mathbf{0}, \sigma_{\text{pos}}^2 \boldsymbol{I}_3). \tag{13}$$

where $\boldsymbol{I}_3$ is the 3×3 identity matrix. For every node pair with at least one non-protected endpoint, we remove an existing edge with probability p_{drop} and insert a missing edge with probability p_{add}. We apply this augmentation before encoding G_{t-1} and before MiDi centering; the clean new block remains the prediction target.

Suboptimal categorical removals. The categorical removal process defined in FLAGG (Cognolato et al., 2026b) starts from a fixed set of block sizes $\mathcal{D} = \{d_1, \ldots, d_l\}$, and computes the optimal combination of sizes to compose the number of nodes of a data graph G, representing the blocks in which G can be decomposed. This is solved optimally with a dynamic programming algorithm, but we observe suboptimal choices during inference. Thus, we augment the training data by sampling suboptimal, and thus longer, removal sequences. The suboptimal choices start from the optimal one

on a data graph G with n nodes, defined as the multiset of block sizes $\mathcal{B} = \{\{b_1, \dots, b_k\}\}$, such that $\sum_{i=1}^{k} b_i = n$, and $b_i \in \mathcal{D}$. Then, we choose a size d from $\mathcal{D}$ which has $c_d \in \{0, \dots, |\mathcal{B}|\}$ occurrences in $\mathcal{B}$, such that $d(c_d + 1) < n$. Then, we compute the optimal decomposition on $n' = n - d(c_d + 1)$ into blocks $\mathcal{B}'$ using the dynamic programming algorithm, and add to $\mathcal{B}'$ the $c_d + 1$ occurrences of d. This gives a new multiset $\mathcal{B}''$ with $\sum_{i=1}^{k'} b'_i = n$, and $|\mathcal{B}''| > |\mathcal{B}|$. This new multiset is chosen with some probability $p_{\text{subopt}} = 0.3$ instead of the optimal one, and the removal sequence is computed from it.

C.2 Exploration and reweighting

Weighted removal sampling. The original FLAGG paper used full removal sequences as training data, i.e., given a batch of graphs, sample one removal sequence per graph, collate all graphs into a single batch, and use that for training. This might lead to showing trivial graphs multiple times at extreme timesteps, i.e., at the almost complete graph or the almost empty graph, and under-representing the intermediate states where the number of graphs obtained by removing nodes is high. Thus, we restate the batch compilation by sampling removal timesteps t instead of full sequences. Given a data graph G, let $q(G_t|G_{t+1})$ be the removal process transition, and $q(G_t|G_T)$ be the marginal distribution of G_t given $G_T = G$, which can be easily sampled from given the definition of the categorical removals (see Cognolato et al., 2026b). T is computed from G as the minimum number of blocks forming G with sizes taken from the fixed set $\mathcal{D}$. Then, let c_t be the number of graphs G_t that can be obtained by removing $T - t$ blocks from G. Then, the weight assigned to timestep t is $w_t = \sqrt{c_t}$, and the probability of sampling t is its normalized weight across T timesteps. Furthermore, given a budget B of graphs obtained with this sampling procedure from a batch of N data graphs, we sample B_i graphs from each data graph proportionally to its total weight $W_i = \sum_{t=1}^{T_i} w_t$, with at least 1 graph assigned to all. This gives a bigger slice of training data in the batch to more complex graphs.

Reweighting MSE with Min-SNR. During preliminary experiments, we observed a drift in generated positions, moving away from the center, and not aligning with the ground truth. After careful inspection of the trajectory taken by coordinates during denoising, we found that at steps close to 0 (i.e., almost clean samples) the model was only required to output small adjustments, but instead produced large updates. After an isolated training on these timesteps, the model produced the correct adjustments. The problem lay in the imbalance of gradient contributions. We noticed a much higher gradient from noisy samples (i.e., with s close to S) than from clean samples (i.e., with s close to 0). Thus, we implemented the Min-SNR reweighting (Hang et al., 2023), which tackles exactly this problem. In our case, we compute weights as $w_\gamma(s) = \min(\text{SNR}(s), \gamma)$, where $\text{SNR}(s) = \alpha_s^2/\sigma_s^2$ is the signal-to-noise ratio at step s, and γ is a hyperparameter that we set to 5 as in the original paper.

D Factorized Distance and Discrete Denoising Diffusion Model (FD4) as Fill Module

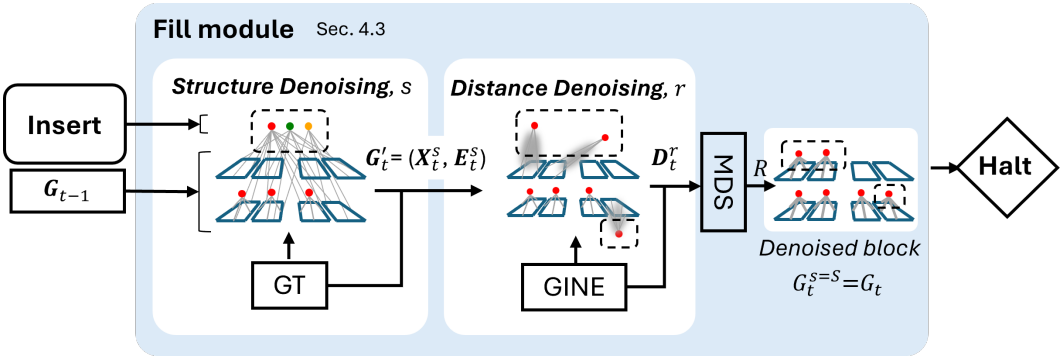

Figure 4: FD4 variant in the Fill module

Table 6: **Method comparison.** The EC baselines are limited to wall/room generation, whereas MiDi+Or, and Ours generate the full hierarchy (floor, building, city).

	Generation					Architecture				
Method	**R**	**W**	**F,B,C**	**Centroids**	**Graph size**	**EC**	**Clustering**	**1-shot**	**Autoreg.**	**Diff.**
EC-□	✓□	✓	✗	✗	✓	✓	✓	✓	✗	✗
EC-L	✓	✓	✗	✓	✓	✓	✓	✓	✗	✗
MiDi+Or	✓	✓	✓	✓	✗	✗	✗	✓	✗	✓
Ours	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓

In this paper we also experiment with another existing one-shot model for 3D graph generation as our Fill module, i.e., Distance and Discrete Denoising Diffusion Model (D4) (Cognolato et al., 2026a). D4 natively samples the graph structure and full distance matrix jointly, then recovers 3D coordinates with Multidimensional Scaling (MDS) (Torgerson, 1952). In a 3DSG, the position of a high-level concept depends on the lower-level concepts connected to it. We therefore factorize the adapted model into two learned stages followed by deterministic reconstruction. We call the factorized model FD4. Figure 4 depicts the relevant block using FD4 instead of MiDi in Fig. 2. First, DiGress (Vignac et al., 2023a) samples $\boldsymbol{X}_t$ and $\boldsymbol{E}_t$, using the same modeling seen for MiDi, without the 3D coordinate path. Merging the sampled topology with G_{t-1} returns an intermediate graph G'_t. Then, a distance model samples the distances on the generated edges. Let $\boldsymbol{D}_t \in \mathbb{R}_+^{|\mathcal{E}_t|-|\mathcal{E}_{t-1}|}$ contain the distances on all new edges. At distance-denoising step r, an edge-regression GINE parameterizes the mean $\mu_\theta(\boldsymbol{D}_t^r, G'_t)$, while Σ_r is the scheduled covariance. The Gaussian reverse transition is

$$p_\theta(\boldsymbol{D}_t^{r-1} \mid \boldsymbol{D}_t^r, G'_t) = \mathcal{N}\big(\mu_\theta(\boldsymbol{D}_t^r, G'_t), \Sigma_r\big). \tag{14}$$

We then recover $\boldsymbol{R}_t$ by minimizing the MDS stress over new-new and new-old edges:

$$\begin{aligned}\boldsymbol{R}_t^* = \arg\min_{\boldsymbol{R}_t} &\sum_{(v,v')\in\mathcal{E}_t\cap(\mathcal{V}'_t\times\mathcal{V}'_t)} \big(\|\boldsymbol{R}_v - \boldsymbol{R}_{v'}\| - \boldsymbol{D}_{(v,v')}\big)^2 + \\ &\sum_{(v,v')\in\mathcal{E}_t\cap(\mathcal{V}'_t\times\mathcal{V}_{t-1})} \big(\|\boldsymbol{R}_v - \rho(v')\| - \boldsymbol{D}_{(v,v')}\big)^2 .\end{aligned} \tag{15}$$

The sampled topology and recovered centroids are merged with G_{t-1} to form G_t.

E Extended baselines

Please find a full summary of the baselines used in our experiments in Table 6. We provide a brief description of each baseline below.

E.1 One-shot generation of 3DSGs

While our autoregressive model covers the task in all its aspects, it is worthwhile to consider current one-shot approaches in graph generation for comparison. One-shot models receive the final size of the graph as input, usually sampled from the empirical distribution of the dataset, and sample the whole connectivity, labels, and features jointly. A caveat encountered with 3DSG generation is how to implement conditioning: only a new portion of the graph needs generation starting from an input graph, and the size of the former depends on the latter. On the graph side, we resolve the issue by inpainting (Lugmayr et al., 2022) the missing nodes, and masking the ground truth plane nodes. For graph sizes, we provide an oracle to the model to make it runnable, i.e., the correct number of nodes is given to the one-shot model. This gives an unfair advantage to the one-shot model compared to our method, but the alternative, i.e., sampling from the empirical distribution, would be an unreliable candidate. We choose MiDi (Vignac et al., 2023b) as the comparison one-shot model.

E.2 Random baseline

For our comparison we also considered a random baseline, serving as a weak reference for the metrics. In the following we describe the sampling procedure. For a dataset, we first compute the training-set empirical distributions of *(i)* number of non-plane nodes $p_{\text{nodes}}(n)$; *(ii)* non-plane labels

Algorithm 1 Fitting and sampling the RAND baseline

Require: Training set $\mathcal{G}_{\text{train}}$; graph of planes $G_{\mathcal{P}} = (\mathcal{V}_{\mathcal{P}}, \mathcal{E}_{\mathcal{P}})$ with centroids ρ
1: Estimate p_{nodes} from $\{|\mathcal{V}(G) \setminus \mathcal{V}_{\mathcal{P}}(G)| : G \in \mathcal{G}_{\text{train}}\}$
2: Estimate p_{labels} from the labels of all non-plane training nodes
3: Estimate p_{edges} as the fraction of candidate training edges with at least one non-plane endpoint that are present
4: Sample $n \sim p_{\text{nodes}}$ and create a set $\mathcal{V}_{\text{new}}$ of n nodes
5: **for all** $v \in \mathcal{V}_{\text{new}}$ **do**
6: Sample $\lambda(v) \sim p_{\text{labels}}$ and $\rho(v) \sim \mathcal{N}(\boldsymbol{\mu}_{\mathcal{P}}, \text{diag}(\boldsymbol{\sigma}^2_{\mathcal{P}}))$
7: **end for**
8: $\mathcal{V} \leftarrow \mathcal{V}_{\mathcal{P}} \cup \mathcal{V}_{\text{new}}$ and $\mathcal{E} \leftarrow \mathcal{E}_{\mathcal{P}}$
9: **for all** $\{u, v\} \subseteq \mathcal{V}$ such that $\{u, v\} \cap \mathcal{V}_{\text{new}} \neq \varnothing$ **do**
10: Insert $\{u, v\}$ into $\mathcal{E}$ with probability p_{edges}
11: **end for**
12: **return** $G = (\mathcal{V}, \mathcal{E}, \lambda, \rho)$

$p_{\text{labels}}(l_e)$; *(iii)* the frequency p_{edges} of edges having a non-plane node on at least one end over all possible edges. Algorithm 1 first samples the extension size from p_{nodes}. For each new node, RAND independently samples a label from p_{labels} and a centroid from $\mathcal{N}(\boldsymbol{\mu}_{\mathcal{P}}, \text{diag}(\boldsymbol{\sigma}^2_{\mathcal{P}}))$, where $\boldsymbol{\mu}_{\mathcal{P}}$ and $\boldsymbol{\sigma}^2_{\mathcal{P}}$ are the coordinate-wise mean and variance of the input planes' centroids. It then inserts every candidate edge with at least one new endpoint independently with probability p_{edges}. Tables 7 and 8 report the results on the synthetic and MSD datasets and on the real dataset, respectively.

Table 7: **Evaluation results for Rand algorithm in synthetic and MSD datasets.** Values are reported $\times 10^2$ as mean $\pm$ standard deviation over three seeds.

Source: Synthetic																							
Method	S-F-□							S-F							S-C								
	FGW ↓	GW↓	W↓	Deg.↓	Spec.↓	Clust.↓	GIN↓	FGW ↓	GW↓	W↓	Deg.↓	Spec.↓	Clust.↓	GIN↓	FGW ↓	GW↓	W↓	Deg.↓	Spec.↓	Clust.↓	GIN↓
RAND	65.0 ± 0.0	50.0 ± 0.0	80.0 ± 0.0	84.0 ± 3.0	55.0 ± 2.0	141.0 ± 4.0	69.0 ± 3.0	69.0 ± 2.0	54.0 ± 2.0	83.0 ± 3.0	104.0 ± 1.0	73.0 ± 1.0	149.0 ± 3.0	74.0 ± 4.0	65.0 ± 1.0	55.0 ± 1.0	75.0 ± 1.0	143.0 ± 0.0	123.0 ± 2.0	160.0 ± 1.0	95.0 ± 2.0
Source: MSD																							
Method	M-F							M-B							M-C								
	FGW ↓	GW↓	W↓	Deg.↓	Spec.↓	Clust.↓	GIN↓	FGW ↓	GW↓	W↓	Deg.↓	Spec.↓	Clust.↓	GIN↓	FGW ↓	GW↓	W↓	Deg.↓	Spec.↓	Clust.↓	GIN↓		
RAND	75.0 ± 0.0	63.0 ± 0.0	86.0 ± 0.0	150.0 ± 1.0	148.0 ± 1.0	180.0 ± 1.0	95.0 ± 2.0	74.0 ± 1.0	64.0 ± 1.0	85.0 ± 1.0	157.0 ± 0.0	183.0 ± 1.0	157.0 ± 1.0	103.0 ± 1.0	75.0 ± 0.0	65.0 ± 0.0	85.0 ± 0.0	166.0 ± 0.0	185.0 ± 0.0	173.0 ± 1.0	111.0 ± 1.0		

Table 8: **Results in real environments for Rand baseline.** Values are reported $\times 10^2$ as mean $\pm$ standard deviation over three seeds.

Source: Real									
Method	R-F			R-B			R-C		
	FGW ↓	GW↓	W↓	FGW ↓	GW↓	W↓	FGW ↓	GW↓	W↓
RAND	105.0 ± 5.0	83.0 ± 3.0	127.0 ± 6.0	81.0 ± 13.0	70.0 ± 9.0	92.0 ± 17.0	92.0 ± 11.0	78.0 ± 7.0	106.0 ± 15.0

F QUALITATIVE RESULTS

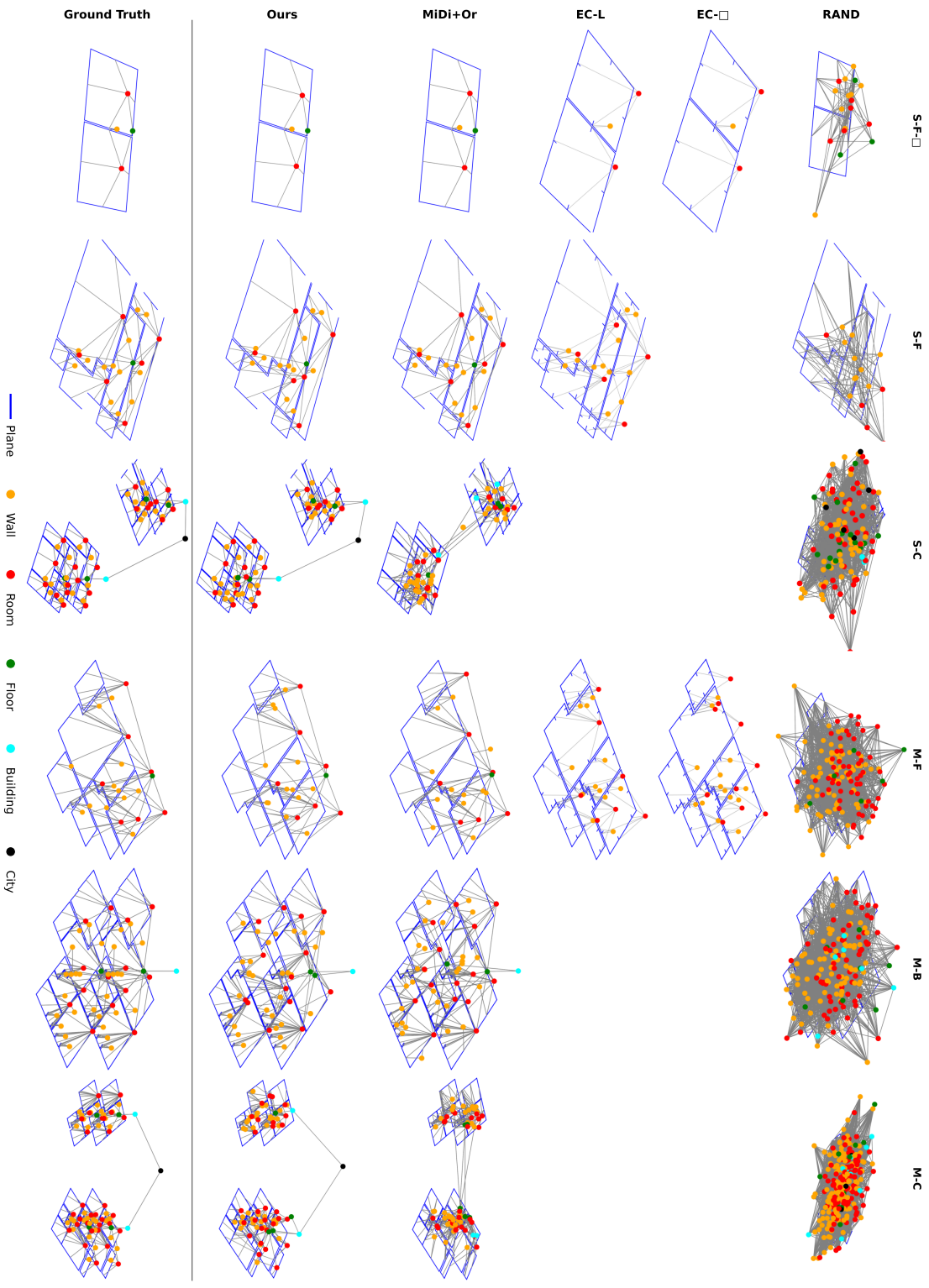

Figure 5: Qualitative results across all training datasets and methods.

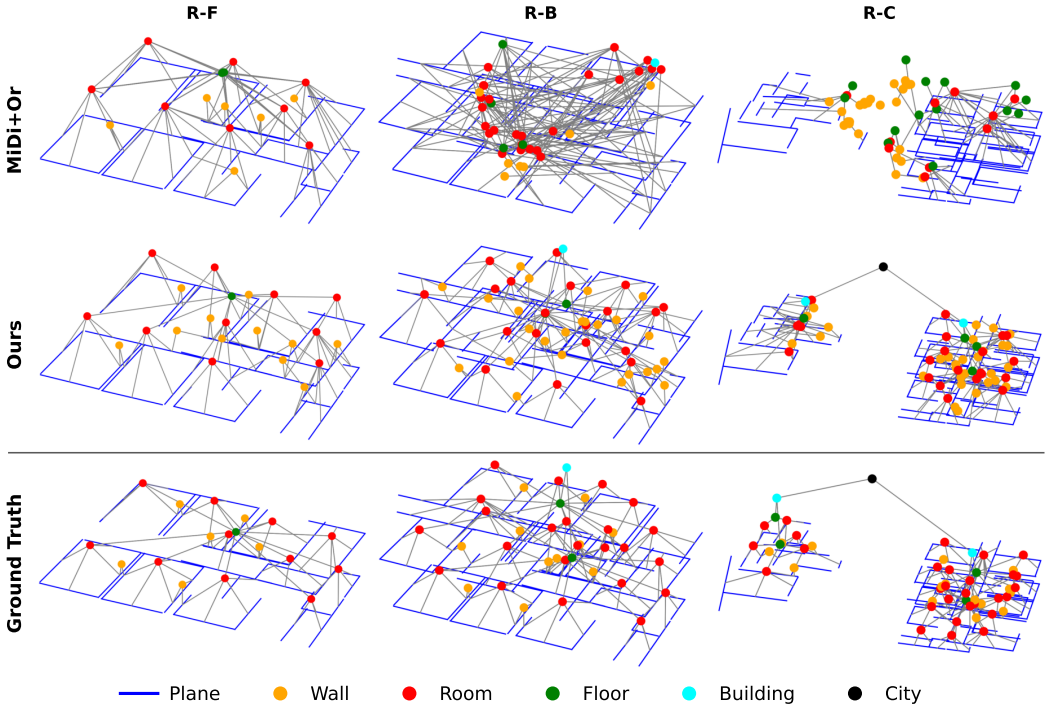

Figure 6: Qualitative results across all real datasets.

G QUANTITATIVE RESULTS

In Tab. 9, we report the sampling time and memory for MIDI+OR and OURS. We kept the original implementation of MiDi with minimal adaptations for MIDI+OR, which breaks down the generation of a batch of graphs into sub-batches, trading sampling speed for memory. Additionally, MIDI+OR has access to the true number of nodes, thus saving memory and computation on potentially overflown nodes. This shows in Tab. 9, as MiDi with oracle has a better memory and time footprint than our model, although Cognolato et al. (2026b) shows the opposite.

Table 9: **Sampling cost on the M- datasets.** Wall-clock time per generated graph and peak GPU memory while generating from the test split (128 / 200 / 128 graphs), on one NVIDIA V100-32GB, mean ± std over 3 seeds. Each method uses its own sampling batch size: OURS generates the whole split in one batch, MIDI+OR in adaptive batches of a few graphs, so peak memory is per configuration, not per graph.

	M-F		M-B		M-C	
Method	Time/graph (s) ↓	Mem. (GiB) ↓	Time/graph (s) ↓	Mem. (GiB) ↓	Time/graph (s) ↓	Mem. (GiB) ↓
MIDI+OR	20.7 ± 0.8	1.43 ± 0.00	38.7 ± 0.5	4.10 ± 0.00	61.4 ± 0.5	3.49 ± 0.00
OURS	27.8 ± 1.1	2.63 ± 0.02	66.8 ± 0.9	6.92 ± 0.04	118.8 ± 0.4	5.97 ± 0.03

H METRICS

Current graph generation metrics are based on the Maximum Mean Discrepancy (MMD) of a pairwise kernel $k(\cdot,\cdot)$ over graph features, extracted by some function f. MMD is defined as:

$$\mathrm{MMD}^2(p,q) = \mathbb{E}_{G,G'\sim p}[k_f(G,G')] + \mathbb{E}_{G,G'\sim q}[k_f(G,G')] - 2\mathbb{E}_{G\sim p,G'\sim q}[k_f(G,G')], \quad (16)$$

which can be empirically computed by replacing the expectations with sample averages. The kernel function we use is the Radial Basis Function (RBF) $k(x,x') = \exp(-\gamma\|x-x'\|^2)$. Then, a metric is identified for each feature extractor: the node degree histogram (Deg.), the clustering coefficient histogram (Clust.), the spectrum (Spec.), and finally, embeddings obtained by a randomly initialized Graph Isomorphism Network (GIN) (Xu et al., 2019).

I EQUIVARIANCE OF THE COMPLETE MODEL

We consider translations and rotations about the vertical axis. Write such a rigid transformation as $g = (\boldsymbol{Q}, \boldsymbol{a})$, where $\boldsymbol{Q}$ is a rotation about $\boldsymbol{e}_z = (0,0,1)^\top$, and let $(g\rho)(v) = \boldsymbol{Q}\rho(v) + \boldsymbol{a}$. Node and edge labels are unchanged, while plane normals, when present, rotate by $\boldsymbol{Q}$.

Relative to the MiDi architecture (Vignac et al., 2023b), our Fill model adds the conditioning graph G_{t-1} and the relative-height feature

$$h_\rho(v,w) = \left|(\rho(v) - \rho(w))^\top \boldsymbol{e}_z\right|. \quad (17)$$

Lemma 1. *Assume that G_{t-1} is nonempty and that Eq. 17 is the only coordinate-dependent scalar added to the MiDi denoiser. Then the conditional Fill kernel is equivariant to translations and rotations about the vertical axis. More precisely, let*

$$G_t \sim p_\theta^F(\cdot \mid m_t, G_{t-1}), \qquad \widetilde{G}_t \sim p_\theta^F(\cdot \mid m_t, g\cdot G_{t-1}). \quad (18)$$

Then

$$\widetilde{G}_t \overset{d}{=} g\cdot G_t, \quad (19)$$

where $\overset{d}{=}$ denotes equality in distribution. Its label and edge distributions are invariant, whereas the generated centroids are equivariant.

Proof. MiDi establishes equivariance of its denoising distribution by centering the input, using rotation-invariant scalar features, and applying isotropic Gaussian noise to the coordinates (Vignac

et al., 2023b). It therefore suffices to check the two modifications used here: conditioning on G_{t-1} and adding Eq. 17.

First, the anchor defined in Section 4.3 is covariant:

$$\boldsymbol{c}_{t-1}(g \cdot G_{t-1}) = \frac{1}{n_{t-1}} \sum_{v \in \mathcal{V}_{t-1}} \big(\boldsymbol{Q}\rho(v) + \boldsymbol{a}\big) = \boldsymbol{Q}\boldsymbol{c}_{t-1}(G_{t-1}) + \boldsymbol{a}. \tag{20}$$

Consequently, centering removes the translation and commutes with $\boldsymbol{Q}$:

$$(g\rho)(v) - \boldsymbol{c}_{t-1}(g \cdot G_{t-1}) = \boldsymbol{Q}\big(\rho(v) - \boldsymbol{c}_{t-1}(G_{t-1})\big). \tag{21}$$

Second, rotations about the vertical axis satisfy $\boldsymbol{Q}^\top \boldsymbol{e}_z = \boldsymbol{e}_z$. Therefore the relative height is unchanged:

$$h_{g\rho}(v, w) = \Big|\big(\boldsymbol{Q}(\rho(v) - \rho(w))\big)^\top \boldsymbol{e}_z\Big| = h_\rho(v, w). \tag{22}$$

Plane-normal inner products, when used, remain invariant because $(\boldsymbol{Q}\boldsymbol{n}_v)^\top(\boldsymbol{Q}\boldsymbol{n}_w) = \boldsymbol{n}_v^\top \boldsymbol{n}_w$. Hence all scalar inputs remain invariant in the centered frame, so the MiDi argument gives invariant label and edge distributions and an equivariant coordinate distribution throughout the reverse process in Eq. 8.

Finally, let $\bar{\rho}_t(v)$ be a sampled centroid in the centered frame. In the transformed run it is distributed as $\boldsymbol{Q}\bar{\rho}_t(v)$. By Eq. 20, restoring the anchor gives

$$\boldsymbol{Q}\bar{\rho}_t(v) + \boldsymbol{c}_{t-1}(g \cdot G_{t-1}) = \boldsymbol{Q}\big(\bar{\rho}_t(v) + \boldsymbol{c}_{t-1}(G_{t-1})\big) + \boldsymbol{a}, \tag{23}$$

which is exactly the action of g. This proves Eq. 19. □

Theorem 1 (Equivariance of the complete generator)**.** *Assume that $G_{\mathcal{P}}$ is nonempty, the Insert and Halt modules use features invariant to translations and rotations about the vertical axis, and Eq. 17 is the only coordinate-dependent scalar added to the MiDi denoiser. For a transformation g described above, let*

$$G_{\textit{final}} \sim p_{\phi,\theta,\nu}(\cdot \mid G_{\mathcal{P}}), \qquad \widetilde{G}_{\textit{final}} \sim p_{\phi,\theta,\nu}(\cdot \mid g \cdot G_{\mathcal{P}}). \tag{24}$$

Then

$$\widetilde{G}_{\textit{final}} \overset{d}{=} g \cdot G_{\textit{final}}, \tag{25}$$

where $\overset{d}{=}$ denotes equality in distribution. Thus the complete FLAGG generator with MiDi as its Fill module is equivariant in distribution under arbitrary translations and rotations about the vertical axis ($\mathrm{SE}(2)_{xy} \times \mathbb{R}_z$). Node and edge labels, insertion counts, and halting decisions are invariant, while centroid coordinates are equivariant and plane normals rotate with $\boldsymbol{Q}$. Equivariance is not guaranteed for rotations that tilt the vertical axis.

Proof of Theorem 1. The Insert kernel $p_\phi^I(m_t \mid G_{t-1})$ and Halt kernel $p_\nu^H(H_t \mid G_t)$ are invariant by assumption. Conditional on the same m_t, the Fill outputs satisfy the equality in distribution in Eq. 19; merging the sampled block with G_{t-1} commutes with g. Induction over t gives the same law for the halting time and Eq. 25 for the terminal graph. □

The result is not, in general, an $\mathrm{SE}(3)$ statement. If $\rho(v) - \rho(w) = \boldsymbol{e}_x$ and $\boldsymbol{Q}$ rotates $\boldsymbol{e}_x$ to $\boldsymbol{e}_z$, the relative height changes from 0 to 1. Thus the vertical feature restricts the guaranteed symmetry to translations and rotations about the vertical axis. Removing this feature would recover the SE(3) symmetry from MiDi.