A. Appendix

A.1. Feature engineering

As explained in §3.1, our indoor scene representation is an attributed graph. We now describe these attributes in detail.

- **Room node features X_R,**
  1. ***Node Type:*** A 4D 1-hot embedding of the node type. The four categories are wall, door, floor, window.
  2. ***Room Type:*** A 4D 1-hot embedding of the room type. The four categories are living room, bedroom, dining room, library.
  3. ***Location:*** A 6D vector representing the minimum and maximum vertex positions of the 3D bounding box corresponding to the room node.
  4. ***Normal:*** A 3D vector representing the direction of the normal vector corresponding to the room node.
- **Furniture node features X_F,**
  1. ***Super category:*** A 7D 1-hot embedding of the node type. The seven categories are Cabinet/Shelf, Bed, Chair, Table, Sofa, Pier/Stool, Lighting.
  2. ***Shape:*** A 1024D embedding of the 3D shape descriptor obtained by processing a 3D point cloud of the furniture item through PointNet.
  3. ***Location:*** A 3D vector representing the centroid of the furniture item.
  4. ***Orientation:*** A 3D vector representing the direction the "front" side of the furniture is facing.
  5. ***Size:*** A 3D vector representing the dimensions of the furniture along each axis.
- **Furniture-furniture edge features X_{FF}**, let us arbitrarily chose a furniture-furniture edge and label its source node as s with receiver node as r. We will describe the features for this edge.
  1. ***Center-center distance:*** A scalar representing the distance between the centroids of furniture s and r.
  2. ***Relative orientation:*** A scalar representing the signed dot product between the orientation feature of s and r.
  3. ***Center-center orientation:*** A 3D vector representing the unit vector connecting the centroids of r and s.
  4. ***Orientation:*** A 3D vector representing the direction the "front" side of the furniture is facing.
  5. ***Bbox-bbox distance:*** The shortest distance between the corners of the bounding box of s and r.
- **Room-furniture edge features X_{RF}**, let us arbitrarily chose a room-furniture edge and label its source node as s with receiver node as r. We will describe the features for this edge.
  1. ***Center-room distance:*** A scalar representing the distance between the centroid of furniture r and room node s. This is computed in 2D as a point to line distance. Every wall/window/door can be treated as a line segment in 2D.
  2. ***Center-room center:*** A scalar representing the distance between the centroids of r and s.
  3. ***Bbox-room dist*** A scalar representing the shortest distance between corners of the bounding box of furniture item r to the room node s. This is computed in 2D as a point-to-line distance.
  4. ***Bbox-room center*** A scalar representing the shortest distance between corners of the bounding box of furniture item r to the centroid of the room node s.
  5. ***Relative orientation:*** A signed dot product between the normal of s with the orientation attribute of r.
- **Room-room edge features X_{RR}**, let us arbitrarily chose a room-room edge and label its source node as s with receiver node as r. We will describe the features for this edge.
  1. ***Center-center distance:*** A scalar representing the distance between the centroids of room nodes s and r.
  2. ***Relative orientation:*** A scalar representing the signed dot product between the normal vectors of s and r.
  3. ***Longest distance:*** A scalar representing the longest distance between the corners of the bounding boxes of r and s.
  4. ***Shortest distance:*** A scalar representing the shortest distance between the corners of the bounding boxes of r and s.

A.2. Architecture Details

A.2.1 Message-Passing Graph Neural Network (MP-GNN)

Here we describe the message-passing mechanism employed in the Graph Neural Networks used in this work. The design of the MP-GNN is inspired by the work of [25], where each layer $l = 1 \ldots, L$ of the MP-GNN maps a graph G^{l-1} to another graph G^l by updating the graph's node and edge features. Specifically, let h_i^l and h_{ij}^l denote the features of node i and edge (i, j) of graph G^l, respectively. Let the input to the network be the graph $G^0 = G$, so that h_i^0 and h_{ij}^0 denote the node features and edge features, respectively. At each iteration of node and edge refinement, the MP-GNN: (1) adapts the scalar edge weights by using an attention mechanism; (2) updates the edge attributes depending on the edge type, the attention-based edge weights, the attributes of the connected nodes and the previous edge attribute; and (3) updates the node attribute by aggregating attributes from incoming edges , as described next.

Computing attention coefficients: At each step l of refinement, we update the scalar edge weights by computing an attention coefficient a_{ij}^l that measures the relevance of node j for node i as follows:

$$\begin{aligned} \gamma_{ij}^l &= \rho(w_a^{\epsilon_{ij}\top}[W_r^{\nu_i} h_i^l; W_s^{\nu_j} h_j^l; W_{rs}^{\epsilon_{ij}} h_{ij}^l] \\ a_{ij}^l &= \frac{\exp(\gamma_{ij}^l)}{\sum_{k \in N_\epsilon(i)} \exp(\gamma_{ik}^l)}, \qquad \forall (i,j) \in E. \end{aligned} \tag{10}$$

In the first equation, each edge (i, j) of $G_l = (V, E, X_l)$ is associated with a score, γ_{ij}^l, obtained by applying a nonlinearity ρ (e.g., a leaky ReLU) to the dot product between a vector of attention weights $w_a^{\epsilon_{ij}}$ for edge type $\epsilon_{ij} \in \{RR, RF, FF\}$, and the concatenation of five vectors: (1) the receiver node feature $h_i^l \in \mathbb{R}^{d_i}$ weighted by a matrix $W_r^{\nu_i}$ for node type $\nu_i \in \{R, F\}$, (2) the sender node feature h_j^l weighted by a matrix $W_s^{v_j}$ for node type v_j, and (3) the receiver-sender edge feature h_{ij}^l weighted by a matrix $W_{rs}^{\epsilon_{ij}}$ for edge type ϵ_{ij}. All weight matrices are also indexed by the GNN layer l which is suppressed to prevent notational clutter.

Updating the node and edge features: At each step l of refinement, we update the edge and node features as:

$$\begin{aligned} h_{ij}^l &= a_{ij}^l (U_{edge}^{\epsilon_{ij}} W_s^{\nu_j} h_j^{l-1} + W_{rs}^{\epsilon_{ij}} h_{ij}^{l-1}), \quad \forall (i,j) \in E \\ h_i^l &= \rho(h_i^{l-1} + \sum_{k \in N_{\epsilon_{ij}}(i)} U_{node}^{\epsilon_{ij}} W_s h_{ik}^l), \qquad \forall i \in V. \end{aligned} \tag{11}$$

The first equation updates the features of edge (i, j) by taking a weighted combination of the features of node j and the features of edge (i, j) in the previous layer, with weight matrices $W_s^{\nu_j}$ and $W_{rs}^{\epsilon_{ij}}$, respectively. The matrix $U_{edge}^{\epsilon_{ij}}$ is a learnable projection matrix to change the dimensionality of the first term (transformed node feature) with that of the second term (transformed edge feature).

The second equation updates the feature of node i as the sum of the feature of node i from the previous layer (a residual connection) and the sum of the features of all edges connected to node i after applying a non-linearity ρ, such as a ReLU or leaky-ReLU. Here, $U_{node}^{\epsilon_{ij}}$ denotes a learnable projection matrix for edge type ϵ_{ij}.

All weight matrices are also indexed by the GNN layer l which is suppressed to prevent notational clutter.

A.2.2 Encoder, Decoder and Room Aggregator networks

We employ three MP-GNNs in this work. One for the VAE encoder, one for the decoder and one for the Room Aggregator component of the prior network. Each of these MP-GNNs are three layered graph neural networks. In Table 2, 3, 4 we summarize the architecture of the Encoder, Decoder and Room Aggregator networks respectively.

The RNN Prior network (second component of our Graph Prior) is implemented by a single layer Gated Recurrent Unit (GRU) with hidden vector of size 64×64 (the latent space is 64D).

A.3. The Reconstruction Loss of the ELBO

The reconstruction loss is the first term in the ELBO (9).

$$\mathbb{E}_{Z \sim q_\phi(Z|n_F, G, T)}[\log p_{\theta'}(G_F \mid Z, n_F, G_R, T)]$$

Table 2. *Encoder architecture. The network parameters are as defined in Section A.2.1*

Encoder			
Network parameters	**Layer 1**	**Layer 2**	**Layer 3**
$W_{r/s}^{R}$	17×17	17×17	17×17
$W_{r/s}^{F}$	256×1034	128×256	64×128
W_{rs}^{FF}	26×6	52×26	6×52
W_{rs}^{RR}	26×4	52×26	6×52
W_{rs}^{RF}	26×5	52×26	6×52
w_{a}^{FF}	1×538	1×308	1×134
w_{a}^{RR}	1×60	1×86	1×40
w_{a}^{RF}	1×299	1×197	1×87
U_{edge}^{FF}	26×256	52×128	6×64
U_{edge}^{RR}	26×17	52×17	6×17
U_{edge}^{RF}	26×17	52×17	6×17
U_{node}^{FF}	256×26	128×52	64×6
U_{node}^{RR}	17×26	17×52	17×6
U_{node}^{RF}	256×26	128×52	64×6

In subsection 3.2 we described the distribution $p_{\theta'}(G_F \mid Z, n_F, G_R, T)$ as

$$\begin{aligned} & p_{\theta'}(G_F \mid Z, n_F, G_R, T) \\ = & \prod_{i=1}^{n_F} p_{\theta'}(shape_i \mid Z, G_R, T) p_{\theta'}(orien_i \mid Z, G_R, T) \\ & p_{\theta'}(loc_i \mid Z, G_R, T) p_{\theta'}(size_i \mid shape_i) p_{\theta'}(cat_i \mid shape_i). \end{aligned}$$

Here $shape_i, orien_i, loc_i, size_i$ and cat_i refer to the ground truth values of the 3D shape descriptor (PointNet features), orientation, location, size and category respectively for furniture node i in G_F.

We will now describe each of these distributions in detail along with the corresponding loss term. For the normal and lognormal distributions used we assume the variance parameter is a fixed constant equal to 1 and do not learn them.

$$\begin{aligned} p_{\theta'}(shape_i \mid Z, G_R, T) &= \mathcal{N}(\mu_{\theta'}^{shape}(Z, G_R, T), const.) \\ \log p_{\theta'}(shape_i \mid Z, G_R, T) &= -\left(\mu_{\theta'}^{shape}(Z, G_R, T) - shape_i\right)^2 \end{aligned}$$

$$\begin{aligned} p_{\theta'}(orient_i \mid Z, G_R, T) &= Categorical(\Theta_{\theta'}^{orient}(Z, G_R, T)) \\ \log p_{\theta'}(orient_i \mid Z, G_R, T) &= -H(orient_i, \Theta_{\theta'}^{orient}(Z, G_R, T)) \end{aligned}$$

Table 3. *Decoder architecture. The network parameters are as defined in Section A.2.1*

Decoder			
Network parameters	**Layer 1**	**Layer 2**	**Layer 3**
$W_{r/s}^{R}$	17×17	17×17	17×17
$W_{r/s}^{F}$	128×64	256×128	1034×256
W_{rs}^{FF}	26×128	52×26	6×52
W_{rs}^{RR}	26×4	52×26	6×52
W_{rs}^{RF}	26×81	52×26	6×52
w_{a}^{FF}	1×282	1×564	1×2074
w_{a}^{RR}	1×60	1×86	1×40
w_{a}^{RF}	1×171	1×325	1×1057
U_{edge}^{FF}	26×128	52×256	6×1034
U_{edge}^{RR}	26×17	52×17	6×17
U_{edge}^{RF}	26×17	52×17	6×17
U_{node}^{FF}	128×26	256×52	1034×6
U_{node}^{RR}	17×26	17×52	17×6
U_{node}^{RF}	128×26	256×52	1034×6

Table 4. *Room Aggregator architecture. This network only processes the room subgraph G_R and hence all the weight matrices associated with furniture nodes are absent. The network parameters are as defined in §A.2.1*

Room Aggregator			
Network parameters	**Layer 1**	**Layer 2**	**Layer 3**
$W_{r/s}^{R}$	32×17	48×32	64×48
W_{rs}^{RR}	26×4	52×26	6×52
w_{a}^{RR}	1×90	1×148	1×134
U_{edge}^{RR}	26×32	52×48	6×64
U_{node}^{RR}	32×26	48×52	64×6

$orient_i$ is a 4D 1-hot embedding of the ground truth orientation and Θ is the parameter of the categorical distribution. $H(.,.)$

denotes the cross entropy between two distributions.

$$p_{\theta'}(loc_i \mid Z, G_R, T) = \mathcal{N}(\mu^{loc}_{\theta'}(Z, G_R, T), const.)$$
$$\log p_{\theta'}(loc_i \mid Z, G_R, T) = -\left(\mu^{loc}_{\theta'}(Z, G_R, T) - loc_i\right)^2$$

$$p_{\theta'}(size_i \mid shape_i) = LogNormal(\mu^{size}_{\theta'}(\mu^{shape}_{\theta'}(Z, G_R, T)), const.)$$
$$\log p_{\theta'}(size_i \mid shape_i) = -\left(\mu^{size}_{\theta'}(\mu^{shape}_{\theta'}(Z, G_R, T)) - \ln size_i\right)^2$$

$$p_{\theta'}(cat_i \mid shape_i) = Categorical(\Theta^{cat}_{\theta'}(\mu^{shape}_{\theta'}(Z, G_R, T)))$$
$$\log p_{\theta'}(cat_i \mid shape_i) = -H(cat_i, \Theta^{cat}_{\theta'}(\mu^{shape}_{\theta'}(Z, G_R, T)))$$

Each furniture node feature in the output of the MP-GNN Decoder is individually processed using a multi-layer Perceptron (MLP). Specifically, let $\hat{h}^L_i$ be the output feature of the decoder for furniture node i. In our experiments $\hat{h}^L_i \in \mathbb{R}^{1034}$.

$$\mu^{shape}_{\theta'}(Z, G_R, T) = \text{Linear}_{\theta'}(1034, 1024)$$

$$\Theta^{orient}_{\theta'}(Z, G_R, T)) = \text{SoftMax}(\text{Linear}_{\theta'}(1034, 4))$$

$$\mu^{loc}_{\theta'}(Z, G_R, T) = \text{Linear}_{\theta'}(1034, 3)$$

$$\mu^{size}_{\theta'} = \text{Linear}_{\theta'}(1024, 512) \rightarrow \text{ ReLU } \rightarrow \text{Linear}_{\theta'}(512, 3)$$

This is a three-layer MLP with ReLU activations and 1024 input neurons since the input is the mean of the shape features $\mu^{shape}_{\theta'}(Z, G_R, T))$.

$$\Theta^{cat}_{\theta'} = \text{Linear}_{\theta'}(1024, 512) \rightarrow \text{ ReLU } \rightarrow \text{Linear}_{\theta'}(512, 7)$$

The output is 7D since there are 7 super-categories in our dataset, Cabinet/Shelf, Bed, Chair, Table, Sofa, Pier/Stool, Lighting.

$\text{Linear}_{\theta'}(x, y)$ denotes a linear layer with x input neurons and y input neurons. Recall θ' denotes all the parameters of the Decoder including the GNN and the output MLPs.

A.4. Computing the KL divergence term

A.4.1 Derivation for (7)

Rewriting (5)

$$\pi^* = \underset{\pi}{\text{argmin}}\, KL(q_\phi(Z \mid n_F, G, T) \mid\mid p_{\theta''}(Z \mid n_F, G_R, T; \pi)) \tag{12}$$

For simplicity let us denote the two distributions as

$$q_\phi(Z \mid n_F, G, T) = \mathcal{N}(\mu_0, \Sigma_0)$$
$$p_{\theta''}(Z \mid n_F, G_R, T; \pi)) = \mathcal{N}(\tilde{\pi}\mu_1, \tilde{\pi}\Sigma_1\tilde{\pi}^T),$$

where $\tilde{\pi} = \pi \otimes I_{d_F \times d_F}$.

The KL divergence between two Gaussian distributions is known analytically,

$$
\begin{aligned}
&KL(q_\phi(Z \mid n_F, G, T) \mid\mid p_{\theta''}(Z \mid n_F, G_R, T; \pi))) \\
&= \frac{1}{2}\left(Tr(\tilde{\pi}\Sigma_1^{-1}\tilde{\pi}^T\Sigma_0) + (\tilde{\pi}\mu_1 - \mu_0)^T\tilde{\pi}\Sigma_1^{-1}\tilde{\pi}^T(\tilde{\pi}\mu_1 - \mu_0)\right) \\
&\quad + \frac{1}{2}\left(\ln\left[\frac{det(\tilde{\pi}\Sigma_1\tilde{\pi}^T)}{det(\Sigma_0)}\right] - n_F d_F\right) \\
&\equiv \left(Tr(\tilde{\pi}\Sigma_1^{-1}\tilde{\pi}^T\Sigma_0) + (\tilde{\pi}\mu_1 - \mu_0)^T\tilde{\pi}\Sigma_1^{-1}\tilde{\pi}^T(\tilde{\pi}\mu_1 - \mu_0)\right) \\
&= Tr(\Sigma_1^{-1}\tilde{\pi}^T\Sigma_0\tilde{\pi}) + \mu_1^T\Sigma^{-1}\mu_1 - \mu_1^T\Sigma_1^{-1}\tilde{\pi}^T\mu_0 \\
&\quad - \mu_0^T\tilde{\pi}\Sigma_1^{-1}\mu_1 + \mu_0^T\tilde{\pi}\Sigma_1^{-1}\tilde{\pi}^T\mu_0 \\
&\equiv Tr(\Sigma_1^{-1}\tilde{\pi}^T\Sigma_0\tilde{\pi}) - \mu_1^T\Sigma_1^{-1}\tilde{\pi}^T\mu_0 - \mu_0^T\tilde{\pi}\Sigma_1^{-1}\mu_1 \\
&\quad + \mu_0^T\tilde{\pi}\Sigma_1^{-1}\tilde{\pi}^T\mu_0 \\
&\equiv Tr(\Sigma_1^{-1}\tilde{\pi}^T\Sigma_0\tilde{\pi}) - 2\mu_1^T\Sigma_1^{-1}\tilde{\pi}^T\mu_0 + \mu_0^T\tilde{\pi}\Sigma_1^{-1}\tilde{\pi}^T\mu_0
\end{aligned}
\tag{13}
$$

The third equivalence is obtained by observing that the constant $n_F d_F$ (which is the dimension of the support of the two Gaussian distributions) does not affect the solution of (12) and that permuting the rows and column of a matrix by the same permutation does not change its determinant and hence the minimizer π^* will also not depend on this term. By similar reasoning, we also ignore the factor of $\frac{1}{2}$. In the fifth equivalence we again ignore terms that don't affect π^*. Using the cyclic property of the $Trace$ operator we can rewrite the last term on the RHS of (13) as

$$
\mu_0^T\tilde{\pi}\Sigma_1^{-1}\tilde{\pi}^T\mu_0 = Tr\left(\mu_0^T\tilde{\pi}\Sigma_1^{-1}\tilde{\pi}^T\mu_0\right) = Tr\left(\Sigma_1^{-1}\tilde{\pi}^T\mu_0\mu_0^T\tilde{\pi}\right)
$$

Substituting this results in (13) we obtain the desired result.

$$
\begin{aligned}
&KL(q_\phi(Z \mid n_F, G, T) \mid\mid p_{\theta''}(Z \mid n_F, G_R, T; \pi))) \\
&\equiv Tr(\Sigma_1^{-1}\tilde{\pi}^T\Sigma_0\tilde{\pi}) - 2\mu_1^T\Sigma_1^{-1}\tilde{\pi}^T\mu_0 + Tr\left(\Sigma_1^{-1}\tilde{\pi}^T\mu_0\mu_0^T\tilde{\pi}\right) \\
&= Tr(\Sigma_1^{-1}\tilde{\pi}^T\left[\Sigma_0 + \mu_0\mu_0^T\right]\tilde{\pi}) - 2\mu_1^T\Sigma_1^{-1}\tilde{\pi}^T\mu_0 \\
&= Tr(\Sigma_1^{-1}\tilde{\pi}^T\left[\Sigma_0 + \mu_0\mu_0^T\right]\tilde{\pi}) - 2Tr\left(\mu_1^T\Sigma_1^{-1}\tilde{\pi}^T\mu_0\right) \\
&= Tr(\Sigma_1^{-1}\tilde{\pi}^T\left[\Sigma_0 + \mu_0\mu_0^T\right]\tilde{\pi}) - 2Tr\left(\tilde{\pi}^T\mu_0\mu_1^T\Sigma_1^{-1}\right)
\end{aligned}
$$

We again used the cyclic property of the $Trace$ operator for the last equality. This concludes our derivation for (7).

A.4.2 Solving (7) using the FAQ approximation

. The FAQ algorithm proceeds as follows,

1. **Choose an initial point:** We choose the uninformative $\pi_0 = \frac{1.1^T}{n_F}$ as the initial estimate for the algorithm. Note, here π^0 is a doubly stochastic matrix and not a permutation matrix.

2. **Find a local solution:** In each iteration i, we linearize the objective at the current iterate π^i

   $$
   \tilde{f}^i(\pi) := f(\pi^i) + Tr\left[\nabla f(\pi^i)^T(\pi - \pi^i)\right]
   $$

   Here, $f(\pi) = Tr\left(\Sigma_1^{-1}\tilde{\pi}\left[\Sigma_0 + \mu_0\mu_0^T\right]\tilde{\pi}^T\right) - 2Tr\left(\tilde{\pi}\mu_0\mu_1^T\Sigma_1^{-1}\right)$. We then solve the following subproblem

   $$
   \begin{aligned}
   &\min_{\pi} Tr(\nabla f(\pi^i)^T(\pi)) \\
   &\text{s.t. } \pi \in D
   \end{aligned}
   \tag{14}
   $$

   Here D is the set of all doubly stochastic matrices. Notice (14) is just a Linear Assignment Problem and can be efficiently solved by the Hungarian Algorithm. Let q^i denote the argmin of (14).

3. **Choose the next iterate:** Finally choose $\pi^{i+1} = \alpha\pi^i + (1-\alpha)q^i$, where $\alpha \in [0,1]$. Here α is chosen by solve the 1D optimization problem analytically.

$$\min_{\alpha} f(\alpha\pi^i + (1-\alpha)q^i) \quad \text{s.t. } \alpha \in [0,1] \tag{15}$$

4. **Repeat steps** 2 **and** 3 **untill convergence.** Steps 2-3 are repeated iteratively untill some termination criteria is met, say $||\pi^i - \pi^{i-1}||_F < \epsilon$. In practice, we do not run steps 2-3 untill convergence but terminate after just 1 iteration of the FAQ algorithm. Running for longer iterations did not result in any significant gains in performance.

5. **Project onto the set of permutation matrices.** Upon termination, the final solution is obtained by projecting π^{final} onto the space of permutation matrices by solving $\min_{\pi \in P} -Tr(\pi^{final}\pi^T)$. Here P is the set of all permutation matrices of size n_F. This is against solved by the Hungarian Algorithm.

In step 2, we need to compute the gradient of $f(\pi^i)$. This can be done analytically. We will start our derivation by stating some facts. First, the gradient of the dot product of two matrices A and X with respect to X is,

$$\frac{d}{dX}\langle A, X\rangle = A \tag{16}$$

Second, for every linear operator M, its adjoint operator $M^\dagger$ is defined such that

$$\langle A, M(X)\rangle = \langle M^\dagger(A), X\rangle \tag{17}$$

Recall,

$$f(\pi) = Tr\left(\Sigma_1^{-1}\tilde{\pi}\left[\Sigma_0 + \mu_0\mu_0^T\right]\tilde{\pi}^T\right) - 2Tr\left(\tilde{\pi}\mu_0\mu_1^T\Sigma_1^{-1}\right)$$

.

We will fist look at the second term,

$$\begin{aligned} Tr\left(\tilde{\pi}\mu_0\mu_1^T\Sigma_1^{-1}\right) &= \langle \Sigma_1^{-1}\mu_1\mu_0^T, M(\pi)\rangle \\ &= \sum_{ij}^{n_F \times n_F} \pi_{ij} Tr\left(\left[\Sigma_1^{-1}\mu_1\mu_0^T\right]_{ij}\right) \\ &= \langle M^\dagger\left(\Sigma_1^{-1}\mu_1\mu_0^T\right), \pi\rangle \end{aligned} \tag{18}$$

Here, $M(\pi) := \pi \otimes I_{d_F \times d_F}$. Given a matrix $A \in \mathbb{R}^{n_F d_F \times n_F d_F}$, we define the operation $[A]_{ij}$ (used in the second equality) as follows. First divide the matrix A into non-overlapping blocks of size $d_F \times d_F$, there are $n_F \times n_F$ such blocks. Now $[A]_{ij}$ denotes the ij^{th} block. In the last equality, we defined the adjoint $L^\dagger(A)$ as $\hat{A}$. Here $\hat{A}$ is a $n_F \times n_F$ matrix obtained from A whose ij^{th} entry is,

$$\hat{A}_{ij} = Tr\left([A]_{ij}\right).$$

.

Combining (18) with (16) we conclude

$$\frac{d}{d\pi} Tr\left(\tilde{\pi}\mu_0\mu_1^T\Sigma_1^{-1}\right) = M^\dagger\left(\Sigma_1^{-1}\mu_1\mu_0^T\right) \tag{19}$$

The gradient of the first term is calculated similarly,

$$\begin{aligned} Tr\left(\Sigma_1^{-1}\tilde{\pi}\left[\Sigma_0 + \mu_0\mu_0^T\right]\tilde{\pi}^T\right) &= \langle \Sigma_1^{-1}\tilde{\pi}\left[\Sigma_0 + \mu_0\mu_0^T\right], M(\pi)\rangle \\ \frac{d}{d\pi} Tr\left(\Sigma_1^{-1}\tilde{\pi}\left[\Sigma_0 + \mu_0\mu_0^T\right]\tilde{\pi}^T\right) &= 2M^\dagger(\Sigma_1^{-1}\tilde{\pi}\left[\Sigma_0 + \mu_0\mu_0^T\right]) \end{aligned} \tag{20}$$

Here $M(\pi)$ is again defined as $\pi \otimes I_{d_F \times d_F}$. Putting it all together,

$$\nabla f(\pi) = 2M^\dagger(\Sigma_1^{-1}\tilde{\pi}\left[\Sigma_0 + \mu_0\mu_0^T\right]) - 2M^\dagger\left(\Sigma_1^{-1}\mu_1\mu_0^T\right) \tag{21}$$

A.5. Learning Under Constraints

Let furniture graph G_F be the input to the encoder and $\tilde{G}_F$ be the furniture graph reconstructed by the decoder. Recall, E_{FF} denotes the edges between furniture nodes in G_F (§3.1) and $\tilde{G}_F$ has the same structure as G_F. We employ the following constraints,

- **furniture-furniture distance constraint**: For every $(v_i, v_j) \in E_{FF}$, we define there relative position as $c_{ij} = ||loc(v_i) - loc(v_j)||_2$, where $loc(v_i)$ denotes the 3D centroid of furniture item v_i from G_F. We define $c_{ij}^{pred} = ||loc^{pred}(v_i) - loc^{pred}(v_j)||_2$ as the relative distance computed from the corresponding location mean prediction by the decoder from $\tilde{G}_F$. Finally we define $d_{FF}^{ij} := MSE(c_{ij}, c_{ij}^{pred})$. Here MSE refers to mean squared error.

- **furniture-room distance constraint**: This constraint restricts the relative position of the predicted furniture items with the room nodes.

  - For the windows and doors, d_{RF}^{ij} is defined analogously as $d_{RF}^{ij} = MSE(c_{ij}, c_{ij}^{pred})$ where $(v_i, v_j) \in E_{RF}$, with the exception that for computing c_{ij}^{pred} we use the ground truth room node location since they are not predicted by the decoder.

  - For walls, c_{ij}^{pred} is computed as the signed distance between the i^{th} wall node and the j^{th} furniture centroid. This ensures the decoder predicts furniture items on the correct side of the wall.

- **furniture-furniture relative orientation constraint**: This constraint enforces the relative orientation between two predicted furniture centroid locations to be close to the ground truth. Specifically we define $\forall (v_i, v_j) \in E_{FF}\ o_{FF}^{ij} = orient(v_i, v_j)^T orient^{pred}(v_i, v_j)$. Here $orient(v_i, v_j)^T$ is the unit vector pointing in the direction $loc(v_i) - loc(v_j)$ computed from G_F. Similarly, $orient^{pred}(v_i, v_j)$ is the unit vector pointing in the direction $loc^{pred}(v_i) - loc^{pred}(v_j)$ computed from the reconstruction $\tilde{G}_F$.

We now present the complete optimization objective as

$$
\begin{aligned}
&\max_{\theta', \theta'', \phi} \mathcal{L}(\theta', \theta'', \phi) \\
&s.t. \quad \frac{1}{n} \sum_{i=1}^{n} \left[\sum_{(v_i, v_j) \in E_{FF}} d_{FF}^{ij} \right] \leq \epsilon \\
&s.t. \quad \frac{1}{n} \sum_{i=1}^{n} \left[\sum_{(v_i, v_j) \in E_{RF}} d_{RF}^{ij} \right] \leq \epsilon \\
&s.t. \quad \frac{1}{n} \sum_{i=1}^{n} \left[\sum_{(v_i, v_j) \in E_{FF}} o_{FF}^{ij} \right] \geq 1 - \epsilon
\end{aligned}
\tag{22}
$$

Here ϵ is a user-defined hyperparameter that determines the strictness of enforcing these constraints[6]

$\mathcal{L}(\theta', \theta'', \phi)$ is as defined in (8) and i is in iterator over the scene graphs in the training set. We employ the learning under constraints framework introduced by [3] which results in a primal-dual saddle point optimization problem. For completeness, we will now describe this algorithm in detail. We begin by explicitly writing out the empirical lagrangian $\hat{\mathcal{L}}_{\theta', \theta'', \phi, \lambda_1, \lambda_2, \lambda_3}$,

[6] One could also have used a different ϵ_i per constraint.

$$
\begin{aligned}
g_1(\theta', \theta'', \phi) &:= \frac{1}{n} \sum_{i=1}^{n} \left[\sum_{(v_i, v_j) \in E_{FF}} d_{FF}^{ij} \right] \\
g_2(\theta', \theta'', \phi) &:= \frac{1}{n} \sum_{i=1}^{n} \left[\sum_{(v_i, v_j) \in E_{RF}} d_{RF}^{ij} \right] \\
g_3(\theta', \theta'', \phi) &:= \frac{1}{n} \sum_{i=1}^{n} \left[\sum_{(v_i, v_j) \in E_{FF}} o_{FF}^{ij} \right] \\
\hat{\mathcal{L}}_{\theta', \theta'', \phi, \lambda_1, \lambda_2, \lambda_3} &:= \mathcal{L}(\theta', \theta'', \phi) + \lambda_3 (1 - \epsilon - g_3(\theta', \theta'', \phi)) \\
&\quad + \lambda_1 (g_1(\theta', \theta'', \phi) - \epsilon) + \lambda_2 (g_2(\theta', \theta'', \phi) - \epsilon)
\end{aligned}
\tag{23}
$$

Here $\lambda_1, \lambda_2, \lambda_3$ are the dual variables. The empirical dual problem is then defined as,

$$
\hat{D}^* := \max_{\lambda_1, \lambda_2, \lambda_3} \min_{\theta', \theta'', \phi} \hat{\mathcal{L}}_{\theta', \theta'', \phi, \lambda_1, \lambda_2, \lambda_3}.
$$

It was shown in [3] that a saddle-point optimization of this empirical dual will give an approximate solution to (22). Algorithm 1 describes the exact steps. At step 5 we require a ρ-optimal minimizer to the empirical Lagrangian, in practice this is done by running the ADAM optimizer for one epoch. After each epoch t, the dual variables are updated depending on the slack evaluated with current parameters $(\theta'^{(t-1)}, \theta''^{(t-1)}, \phi^{(t-1)})$. At an intuitive level, the algorithm is similar to regularized optimization with adaptive Lagrange multipliers (the dual variables). In (23) each term after $\mathcal{L}(\theta', \theta'', \phi)$ can be thought of a regularizer (one corresponding to each constraint). The Langrange multipliers are updated in each epoch to enforce or relax the regularizer depending on whether the corresponding constraint is violated or satisfied.

Algorithm 1 Learning under constraints

Require: *Initializations:* $\theta'^{(0)}, \theta''^{(0)}, \phi^{(0)}$; *Learning rate for dual variables* η; *Number of steps* T

1: $\lambda_1^{(0)} = 0$
2: $\lambda_2^{(0)} = 0$
3: $\lambda_3^{(0)} = 0$
4: **for** $t = 1, \ldots, T$ **do**
5: Obtain $\theta'^{(t-1)}, \theta''^{(t-1)}, \phi^{(t-1)}$ such that,

$$
\hat{\mathcal{L}}_{\theta', \theta'', \phi, \lambda_1, \lambda_2, \lambda_3} \leq \min_{\theta', \theta'', \phi} \hat{\mathcal{L}}_{\theta', \theta'', \phi, \lambda_1, \lambda_2, \lambda_3} + \rho.
$$

6: Update dual variables

$$
\begin{aligned}
\lambda_1^{(t)} &= \left[\lambda_1^{(t-1)} + \eta (g_1(\theta'^{(t-1)}, \theta''^{(t-1)}, \phi^{(t-1)}) - \epsilon) \right]_+ \\
\lambda_2^{(t)} &= \left[\lambda_2^{(t-1)} + \eta (g_2(\theta'^{(t-1)}, \theta''^{(t-1)}, \phi^{(t-1)}) - \epsilon) \right]_+ \\
\lambda_3^{(t)} &= \left[\lambda_3^{(t-1)} + \eta (1 - \epsilon - g_3(\theta'^{(t-1)}, \theta''^{(t-1)}, \phi^{(t-1)})) \right]_+
\end{aligned}
$$

7: **end for**

In Figure 5, we show training curves for ELBO objective (9) and the ELBO objective with constraints (22) which clearly show the benefit of using constraints. The network performance without any constraints is comparable, to that with constraints, for the furniture category, shape and size loss terms. These are arguably much easier to learn than the orientation and position in 3D.

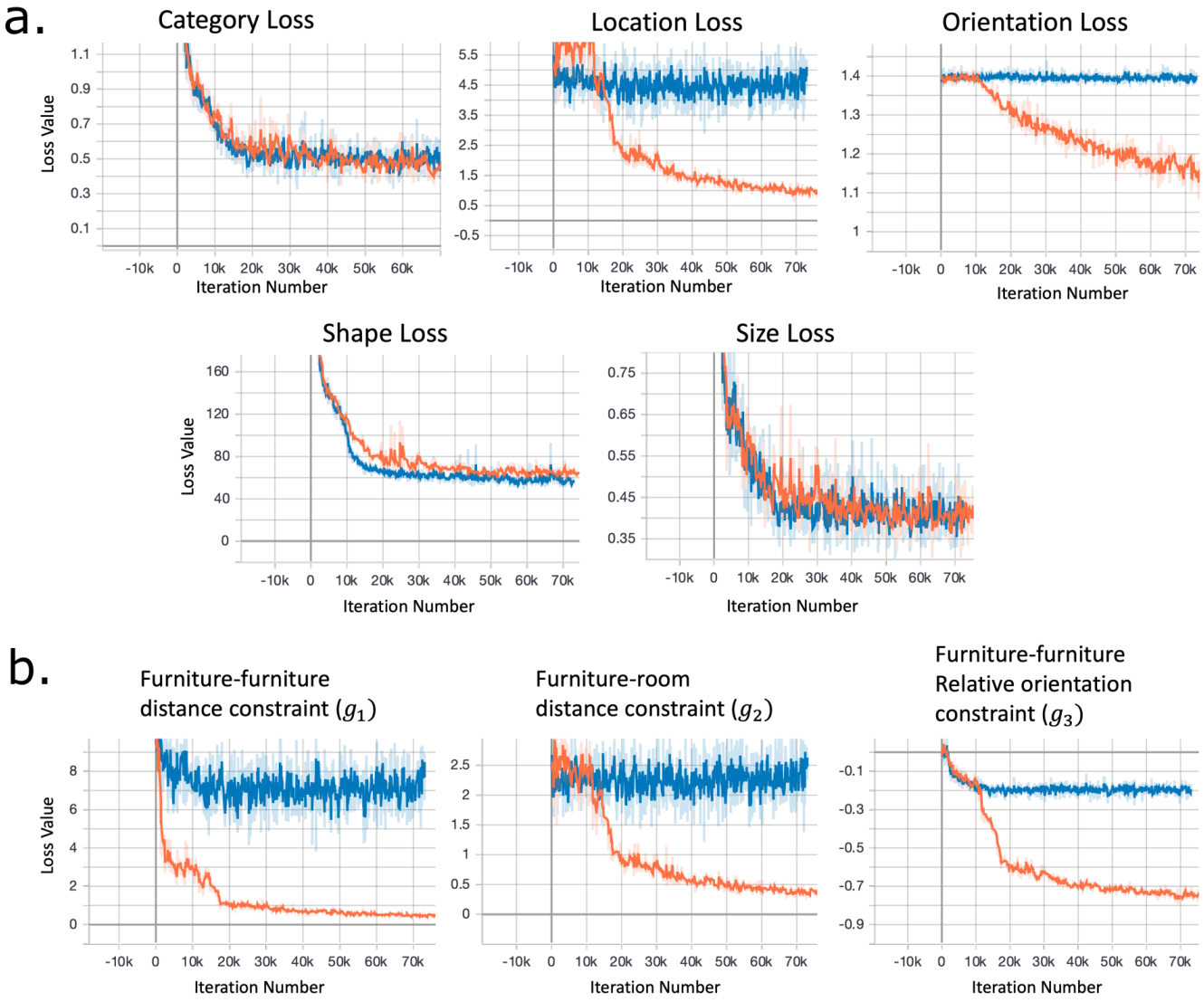

Figure 5. ***Ablation studies showing the effect of constraints on training. The orange curve indicates learning under constraints whereas the blue curve indicates learning without any constraints****(a) Training curves for the terms in the reconstruction loss of the ELBO (first term in (9); (b) Objective values of the constraints as training progresses. Note the x-axis is the number of iterations rather than epoch (one iteration is one batch processed).*

A.6. Diverse furniture layout recommendations for the same room layout

More examples for Figure 1b in Figure 6

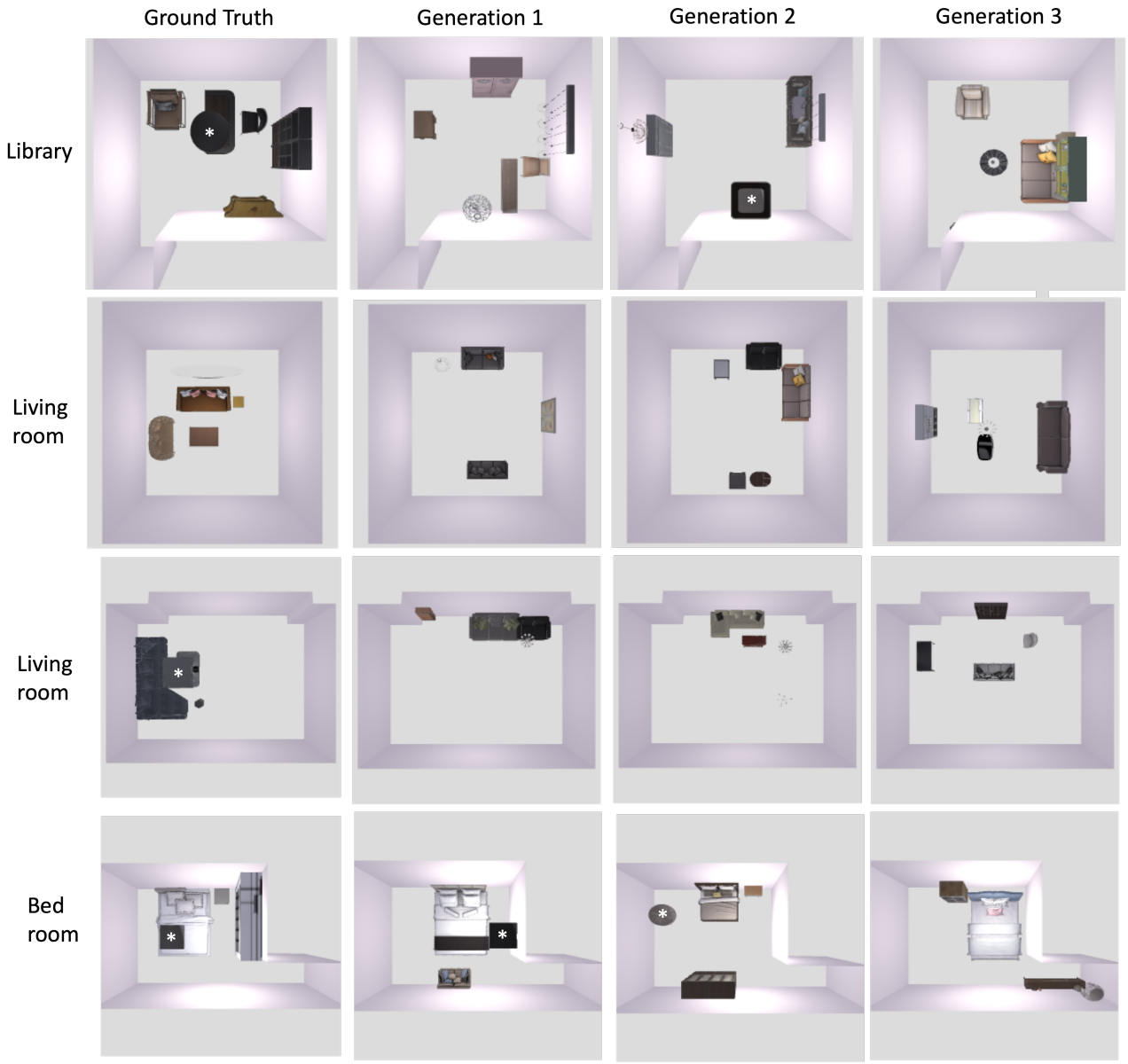

Figure 6. ***Diverse furniture layout recommendations for the same room layout.*** *Each row depicts a specific floor-plan, row 1: library; row 2, 3: living room, row 4: bedroom. The first column is the ground truth design from the test set. The remaining columns are recommendations made by our proposed model. All rooms are top-down rendering of the scene. We mark ceiling lamps with a white asterisk in images where we believe it is hard to recognize from our top-down rendering. In row 1, column 4, the green furniture on top of the sofa is a overhead cabinet.*

A.7. Manipulating Latent Space for Design Recommendations from Database.

In this subsection, we show more Examples for Figure 3 in Figure 7. The procedure is as follows,

1. Create a Database:

    - Iterate over the training set.

- For each scene, pass it through the GNN encoder to obtain the latent code. Store both the scene and the corresponding latent code in the database.

2. Retrieving closest matched scenes

   - Given an empty room layout $\tilde{G_R}$, we iterate over each scene i in the database.
   - For every scene i, the corresponding latent $Z_i = \{Z_i^1, \dots Z_i^{n_{F_i}}\}$ where n_{F_i} is the number of furniture items in scene i.
   - Since Z_i was sampled using our approximate posterior $q_\phi(Z \mid G_i, T_i, n_{F_i})$ we solve (5) using the FAQ algorithm and the Graph prior $p_{\theta''}(Z \mid \tilde{G_R}, \pi, T_i, n_{F_i})$[7] to find the optimal ordering π^*. Here $\tilde{G_R}$ is the given empty room layout.
   - We then evaluate the likelihood of Z_i under our prior and optimal ordering π^*
   - Finally we choose the top 3 scenes which have the highest likelihood under the prior as the closest "match". In other words, these latent codes are very likely under the prior for room layout $\tilde{G_R}$ and thus would result in good designs when passed through our GNN decoder.

In Figure 3 we showed multiple retrieval results for a library, here we give two more examples a bedroom and a living room. Notice that the retrieved scenes can have very different room shape compared to $\tilde{G_R}$ however the furniture arrangements can still look plausible in $\tilde{G_R}$.

A.8. Qualitative comparison of our method with ATISS and baselines

In this subsection, we show more Examples for Figure 2 in Figure 8.

A.9. More results for scene editing

In this subsection, we show more Examples for Figure 4 in Figure 9.

[7] Recall, given any ordering of the latent variables, π, the Graph Prior simulates and auto-regressive model based on π. See (4)

A.10. Analyzing matched furniture nodes for the trained autoregressive prior

In this subsection we analyze the following question - After training, what is the category of the furniture's latent, taken from an encoded scene using the GNN encoder, that gets matched to the first Z^1 sampled using our autoregressive prior?

To answer this we carry out the following steps,

- Iterate over the scenes in the test set.
- Ever scene is a tuple consisting of the attributed scene graph, the room type and the number of furniture's in the room (G, T, n_F). For each scene,
  - Pass it through the trained GNN encoder to obtain the parameters for $q_\phi(Z \mid n_F, G, T)$. Here $Z := \{Z^1, Z^2, \dots Z^{n_F}\}$ is the set of latent variables corresponding to the n_F furniture items to be placed in the room.
  - Next solve (5) using the FAQ algorithm and the Graph prior $p_{\theta''}(Z \mid G_R, \pi, T, n_F)$[8] to find the optimal ordering π^*. Here G_R refers to the room layout graph derived from the input scene graph.
  - This π^* is the optimal assignment between Z^i's obtained from the approximate posterior $q_\phi(Z \mid n_F, G, T)$ and the sequential latent nodes sampled using the auto-regressive prior (See (3)).
- Since each Z^i corresponds to a furniture item, we analyze the frequencies (over the test set) with which a certain category is mapped to the first latent sampled by the auto-regressive prior. This can give insights into what the model has learnt and whether the prior's modelling of furniture placement in indoor scenes correlates with how interior designers begin planning room layouts. In Table 5 we present these results. We found that the first bedroom item matched by our model is a nightstand/bed/light with probability 0.91 and the first living room item matched is a coffee-table/sofa/light with probability 0.76. This is interesting since human designers often start planning with these items.

Table 5. *Frequency of categories mapped to the first latent sampled by the auto-regressive prior*

Categories	**Bedroom**	**Livingroom**
Bed	0.30	0.0
Light	0.19	0.19
Night-stand	0.43	0.0
Chair	0.02	0.11
Sofa	0.01	0.20
Table	0.04	0.10
Pier	0.01	0.01
Coffee-Table	0.0	0.39

[8] Recall, given any ordering of the latent variables, π, the Graph Prior simulates and auto-regressive model based on π. See (4)

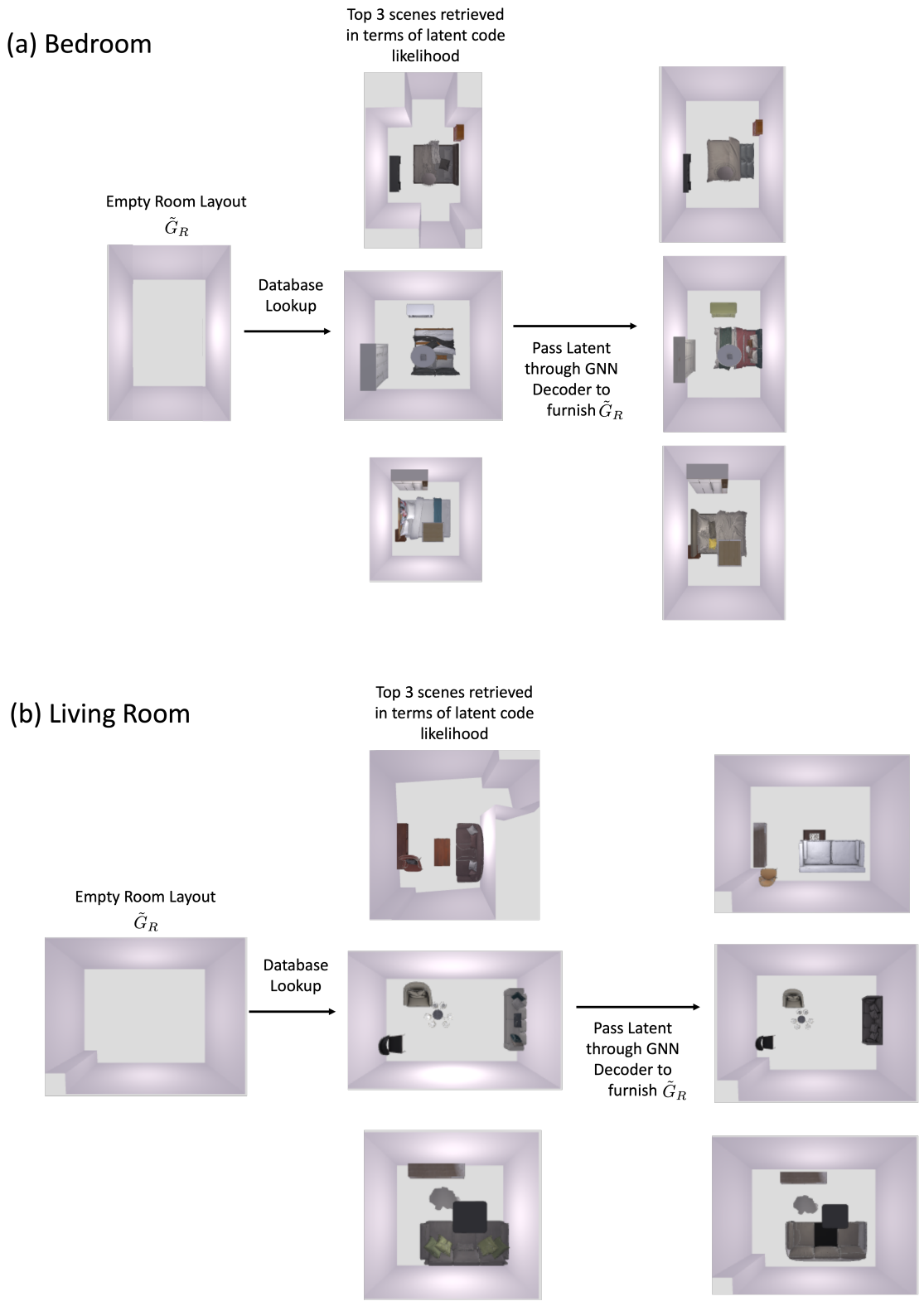

Figure 7. ***Manipulating Latent Space for Design Recommendations from Database.*** *We show results for two room types (a) bedroom and (b) living room. The GNN decoder ensures the synthesized design in $\tilde{G}_R$ has approximately the same spatial arrangement as the design of the retrieved scene. However in some cases (middle room in (a) and top room in (b), the bed and sofa are rotated respectively to ensure the satisfaction of constraints levied by the room layout, for example, sofa should be parallel to the closest wall.*

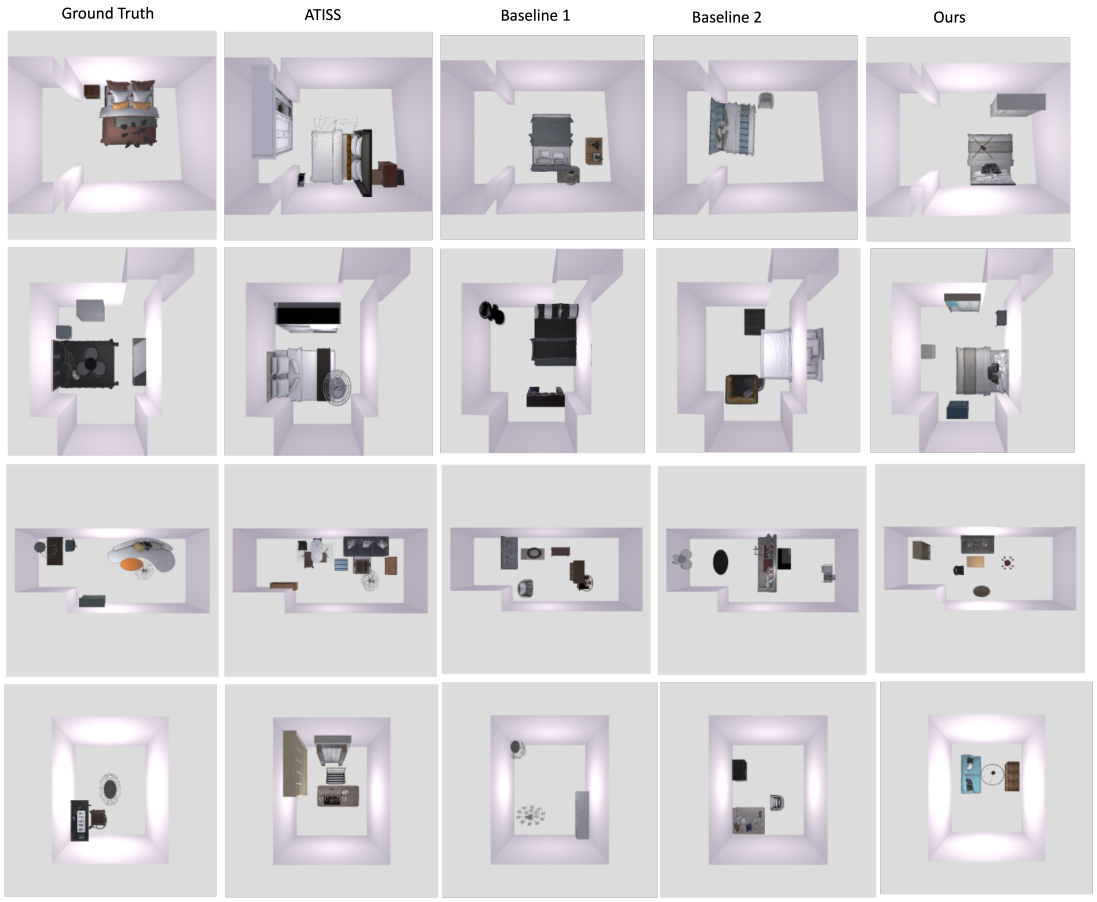

Figure 8. *More results for Qualitative comparison of our method with ATISS and baselines. Row 1,2 are bedrooms. Row 3 is a living room and Row 4 is a library.*

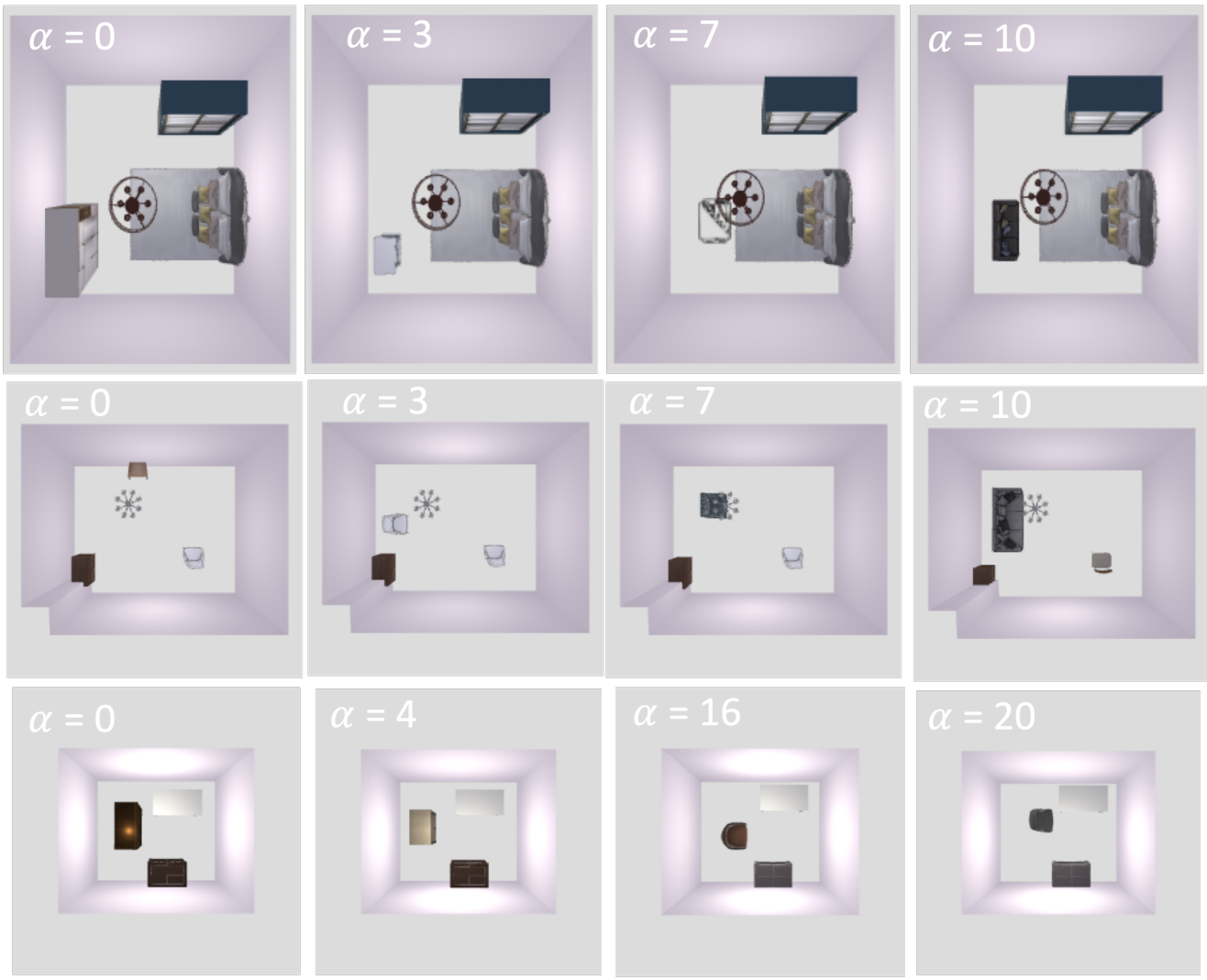

Figure 9. ***Scene Editing.*** *Col 1 is a scene generated by our model. In row 1, we morph the bottom-left cabinet into a sofa by changing the α parameter as explained in text (§4). In row 2 we morph the yellow chair in the top-center into a sofa. In row 3, we morph the cabinet (marked with a yellow spot) into a chair.*