MaskPLAN: Masked Generative Layout Planning from Partial Input

Supplementary Material

This supplementary material is structured as follows:

- Sec. 1 explains the architecture of the Attribute Discrete Latent Model (ADLM).
- Sec. 2 gives definitions of the ADLM loss functions.
- Sec. 3 talks about the framework detail in Graph structured Dynamic Masked AutoEncoders (GDMAE).
- Sec. 4 exhibits more ablation studies.
- Sec. 5 illustrates additional results: (1) qualitative evaluations of our methods and the baselines. (2) flexibility of User-AI interactions in MaskPLAN.

1. Architecture of ADLM

Training on multiple sequences of high-resolution images is a computationally intensive task. The Attribute Discrete Latent Model (ADLM) is leveraged to encode the image information of site condition B, as well as layout attributes C_{img}, S_{img} and R_{img} into a lower dimension as visual tokens $I \in \mathbb{R}^K$, which subsequently are used as pretrained prior to the GDMAE. This model is generally built on VQ-VAE [5], which contains an encoder $\mathcal{E}$ and a decoder $\mathcal{D}$, with a latent embedding space $d \in \mathbb{R}^{K \times V}$. ADLM leveraged a vector quantizer q that indexes the visual tokens $I \in \mathbb{R}^{8 \times 8}$ to map d from the encoder output. The discrete visual tokens I, parsed from the original images, subsequently serve as input and ground truth for training GDMAE.

All image datasets are normalized in the range (0,1), also split into 0.8:0.1:0.1 for training, validating, and testing. We set the size of discrete latent space K to 8×8 and the dimension of the embedding vector V to 64. See Sec. 4 for the ablation study on these hyperparameters.

Module	Layers	Output Size
Encoder $\mathcal{E}$	*Conv2D*(64, s = 2)	$64 \times 64 \times 64$
	Conv2D(128, s = 2)	$32 \times 32 \times 128$
	Conv2D(256, s = 2)	$16 \times 16 \times 256$
	Conv2D(512, s = 2)	$8 \times 8 \times 512$
	Conv2D(64, s = 1)	$8 \times 8 \times 64$
Decoder $\mathcal{D}$	*Conv2DT*(512, s = 1)	$8 \times 8 \times 512$
	Conv2DT(256, s = 2)	$16 \times 16 \times 256$
	Conv2DT(128, s = 2)	$32 \times 32 \times 128$
	Conv2DT(64, s = 2)	$64 \times 64 \times 64$
	Conv2DT(4, s = 2)	$128 \times 128 \times 4$

Table 1. Architecture details of $\mathcal{E}$ and $\mathcal{D}$ in ADLM.

Table 1 describes the architecture in $\mathcal{E}$ and $\mathcal{D}$, where *Conv2D*() and *Conv2DT*() denote 2D convolution layer and 2D transposed convolution layer, respectively. All the layers are configured with 2×2 kernel size and zeros padding.

We trained ADLM using Adam as the optimizer, with an initial learning rate of $10e^{-4}$ and 0.5 decay per every 5 epochs. A batch size of 32 was used for training each of the four pretrained models (B, C, S, and R), and the training process was carried out for 50 epochs. The training duration for all four models on AMD 5950X and a single RTX-Titan GPU was approximately 34 hours, while the time cost for preprocessing these four image datasets (2.23 M images in total) into visual tokens was nearly 41 hours.

2. ADLM Loss Functions

The loss function of ADLM consists of three components: reconstruction loss, Vector Quantisation (VQ) loss, and commitment loss. The VQ loss is leveraged to move the embedding vectors toward the encoder outputs $z_e(M)$ (second term in Eq. 1), and the commitment loss serves to constrain the encoder to commit to one embedding space (third term in Eq. 1). The total loss $\mathcal{L}_{ADLM}$ is structured as follows:

$$\begin{aligned}\mathcal{L}_{ADLM} = Log\mathcal{P}(\boldsymbol{M} \,|\, z_q(M)) + ||\, \boldsymbol{sg}\,[z_e(M)] - d\,||_2^2 \\ + ||\, z_e(M)] - \boldsymbol{sg}\,[d]\,||_2^2\end{aligned} \tag{1}$$

where M refers to the training image, $Log\mathcal{P}(M \,|\, z_q(M))$ is defined as reconstruction loss, $z_q(M))$ and $z_e(M))$ refer to decoder input and encoder output respectively. $\boldsymbol{sg}$ denotes the stopgradient operator, defined as an identity to constrain its operand to be a non-updated constant.

3. GDMAE

Model Architecture. Our GDMAE consists of six components, including the partial input encoder $\mathcal{E}_U$ and five mutually related generators $\mathcal{G}_T$, $\mathcal{G}_C$, $\mathcal{G}_A$, $\mathcal{G}_S$, and $\mathcal{G}_R$. Layout attributes are decoded in a procedural manner, where each prior feature conditions the subsequent ones.

The framework detail of Graph-structured Dynamic Masked Autoencoder (GDMAE) is provided in Table 2: (1) *GlobalEncoder*() denotes the transformer encoder built for partial input. (2) *GeneraEncoder*() and *GeneraDecoder*() correspond to the generator encoder and decoder modules that are explained in the main paper. (3) *PosiEmbed*() represents the positional embedding layer. (4) *MLP*(*Concat*()) indicates the layer combination of *Concatenation* and *MLP*. (5) *Softmax*() refers to the layer combination of *Dropout*, *MLP*, and *Softmax*. (6) $\mathcal{M}()$ represents the masked input sequence. (7) *In*() denotes the input sequence with the [Start] and [End] tokens. The dimensions K and V are inherited from the hyperparameters in the pretrained ADLM (Sec. 1).

	Index	Inputs	Layers	Output Size
$\mathcal{E}_U$	0	-	$\mathcal{M}(Type)$	10×1
	1	-	$\mathcal{M}(Location)$	$10 \times K$
	2	-	$\mathcal{M}(Adjacency)$	10×8
	3	-	$\mathcal{M}(Area)$	$10 \times K$
	4	-	$\mathcal{M}(Region)$	$10 \times K$
	5	-	$In(Site)$	$10 \times K$
	6	0	*PosiEmbed*()	$10 \times H$
	7	1	*PosiEmbed*()	$10 \times H$
	8	2	*PosiEmbed*()	$10 \times H$
	9	3	*PosiEmbed*()	$10 \times H$
	10	4	*PosiEmbed*()	$10 \times H$
	11	5	*MLP*()	$10 \times H$
	12	6-11	*MLP*(*Concat*())	$10 \times H$
	13	12	*GlobalEncoder*()	$10 \times H$
$\mathcal{G}_T$	14	-	$In(Type)$	10×1
	15	14	*PosiEmbed*()	$10 \times H$
	16	13,15	*GeneraDecoder*()	$10 \times H$
	17	16	*Softmax*()	10×8
$\mathcal{G}_C$	18	-	$In(Location)$	$10 \times K$
	19	18	*PosiEmbed*()	$10 \times H$
	20	13,16	*MLP*(*Concat*())	$10 \times H$
	21	20	*GeneraEncoder*()	$10 \times H$
	22	19,21	*GeneraDecoder*()	$10 \times H$
	23	22	*Softmax*()	$10 \times K \times V$
$\mathcal{G}_A$	24	-	$In(Adjacency)$	10×8
	25	24	*PosiEmbed*()	$10 \times H$
	26	20,22	*MLP*(*Concat*())	$10 \times H$
	27	26	*GeneraEncoder*()	$10 \times H$
	28	25,27	*GeneraDecoder*()	$10 \times H$
	29	22	*Softmax*()	$10 \times 8 \times 2$
$\mathcal{G}_S$	30	-	$In(Size)$	$10 \times K$
	31	30	*PosiEmbed*()	$10 \times H$
	32	26,28	*MLP*(*Concat*())	$10 \times H$
	33	32	*GeneraEncoder*()	$10 \times H$
	34	31,33	*GeneraDecoder*()	$10 \times H$
	35	34	*Softmax*()	$10 \times K \times V$
$\mathcal{G}_R$	36	-	$In(Region)$	$10 \times K$
	37	36	*PosiEmbed*()	$10 \times H$
	38	32,34	*MLP*(*Concat*())	$10 \times H$
	39	38	*GeneraEncoder*()	$10 \times H$
	40	37,39	*GeneraDecoder*()	$10 \times H$
	41	40	*Softmax*()	$10 \times K \times V$

Table 2. General architecture of GDMAE in MaskPLAN.

Specifically, our hyperparameter settings follow the ViT-Base [1], where all encoders and decoders are defined with 12 layers of attention modules and 12 heads in multi-head attention. The embedding hidden dimension *H* is set to 768, and the *MLP* size is 3072. Both modules, $\mathcal{G}_G$ and $\mathcal{G}_{S,R}$, are trained with a batch size of 32 for 200 epochs using a linear learning rate warmup and decay.

ViTs	Params	Training *t*	Infer *t*	fid_{img}
ViT-Tiny	175.5 M	34 *h*	2.17s	5.391
ViT-Base	542.1 M	127 *h*	3.61s	4.182
ViT-Large	1203.9 M	429 *h*	6.79s	4.073

Table 3. The training cost and inference performance of MaskPLAN structured with different variants of the ViT model. Quantitative evaluation is calculated on fid_{img}.

ViT Variants. As shown in Table 3, we conducted experiments on MaskPLAN with three variants of the ViT models: ViT-Tiny, ViT-Base, and ViT-Large. Training and inference were performed on an AMD 5950X and a single RTX-Titan GPU. As $\mathcal{G}_G$ and $\mathcal{G}_{S,R}$ are trained and inferred separately, these two modules sum up the final time cost. Notably, MaskPLAN with ViT-Tiny requires the least amount of time for training and inference, but its performance on layout reconstruction is comparatively worse. MaskPLAN with ViT-Large demands significantly more time for both training and inference, resulting in only marginal improvements in layout prediction. MaskPLAN with ViT-Base strikes the optimal balance between time cost and layout reconstruction performance.

Post-Processing. An initial rough layout can be generated by assigning the corresponding room types T when obtaining the final regions R. We further perform a post-processing step to properly align rooms with the site boundary and other rooms. Specifically, each room region in pixel space is converted into a bounding box, represented by corners $p \in \{X_{min}, Y_{min}, X_{max}, Y_{max}\}$, where X and Y denote the left and right corners and min and max refer to the lower and upper corners. These corner points then assess the neighboring environment and calculate the adjustment distance required to match the context, defined as follows:

$$d(R_p, R_t) = \begin{cases} 0 & , \quad if R_p \cap R_t == 0 \\ min_{q \in R_t} ||p - q|| & , \quad otherwise \end{cases} \tag{2}$$

where R_p represents an 9×9 bounding box region centered on the corner point p, R_t denotes the target region to align with, and $q \in R_t$ indicates all the pixel points within the target region. By calculating the minimum distance between the corner points p and the target region R_B, denoted as $min_{q \in R_t} ||p - q||$, each corner in $\{X_{min}, Y_{min}, X_{max}, Y_{max}\}$ can be properly adjusted. We first align point p with the site boundary region B and then realign p with all the existing predicted room regions R.

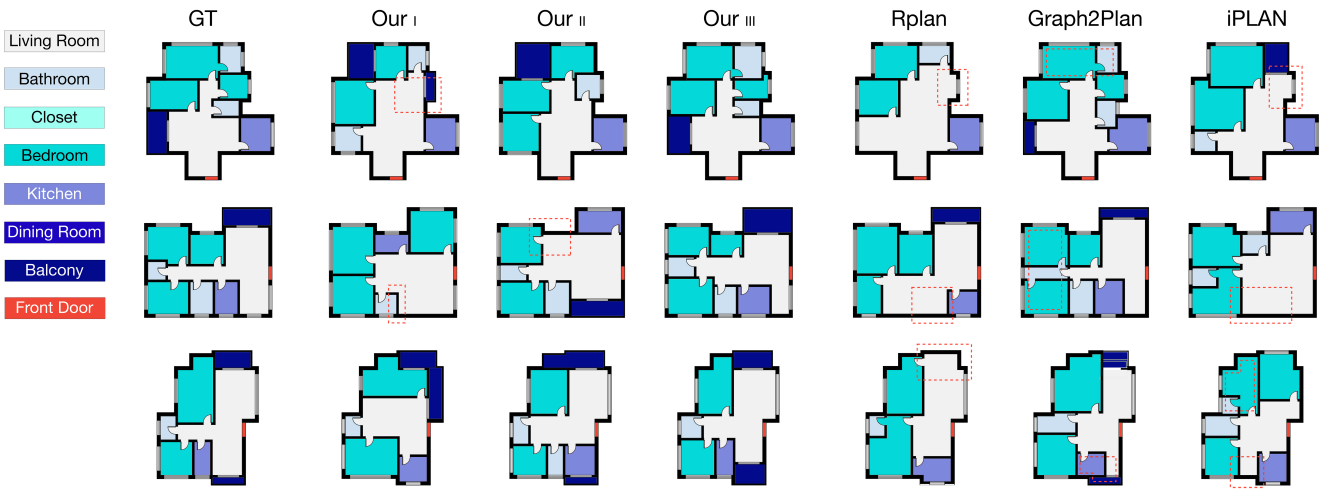

Figure 1. Additional qualitative comparisons on layout reconstruction, with Rplan, Graph2Plan, iPLAN, and three of our approaches.

4. Additional Ablation Studies

We conduct additional ablation studies to demonstrate our technical contribution:

Ablation Setting	fid_{img}
w/o adjacency A	7.215
w/o size S	6.737
w adjacency A and size S	**4.182**

Table 4. Ablation study on the model without training A or training S. Quantitative evaluation is calculated on fid_{img}.

(1) **Attributes A and S**: every floorplan is represented as 5 essential layout attributes $L = \{T, C, A, S, R\}$. We observed several existing literature decompose the floorplan without adjacency A or room size S, leading to the inability to adjust room relations [2, 6] or the absence of manipulation for room sizes [2–4, 6]. The lack of capability in sufficient attribute manipulation may hinder design customization in coping with various demands from users. In addition, we evaluate our model trained without A or S, as shown in Tab. 4. It's evident by the results in fid_{img} that integrating information of A or S could also effectively enhance the accuracy of layout reconstruction.

KV	32	64	128	256
4×4	9.433	7.719	6.513	5.747
8×8	4.394	**4.231**	4.443	4.826
16×16	4.915	5.413	6.291	7.104

Table 5. Ablation study with hyperparameters on latent embedding space $d \in \mathbb{R}^{K \times V}$. Quantitative evaluation is calculated on fid_{img}.

(2) **Latent embedding space**: the latent vector $d \in \mathbb{R}^{K \times V}$ is defined in pretrained ADLM, where K is the size of discrete latent space and V is the dimension of each latent embedding vector. The dimension of visual tokens $I \in \mathbb{R}^{K}$ will correspond to this parameter, serving as the information bridge between ADLM and GMAE. Achieving a careful balance is essential; a larger dimension may result in a lower loss in the pretrained ADLM but poses more challenges for the GMAE during training. Conversely, a smaller dimension makes training in GMAE easier but significantly hinders proper reconstruction in ADLM. As demonstrated in Tab. 5, due to the time-consuming training process on the entire dataset (200+ h), we randomly sampled 25 % training data and conducted experiments with various combinations of K and V. It turns out that $K = 8 \times 8$ and $V = 64$ represent the optimal balance between training these two models.

5. Additional Results

(1) Fig. 1 presents additional comparisons of generated layouts. Similar to the layout comparisons in the main paper, *Our III* demonstrates the best performance compared to all other methods. *Our I* showcases significant diversity in layout creation, while with 25% partial input, most of the rooms in *Our II* align much closer to the ground truth, emphasizing the effectiveness of our partial input guided generation in layout reconstruction.

(2) Fig. 2 illustrates examples of generated results across various ratios of partial input. As indicated in Table 2 of the main paper, when MaskPLAN is provided simply with the boundary condition B, it struggles to reconstruct the ground truth accurately. However, a notable improvement is observed when partial input is introduced.

(3) Single-attribute Guidance. Owing to the versatility of

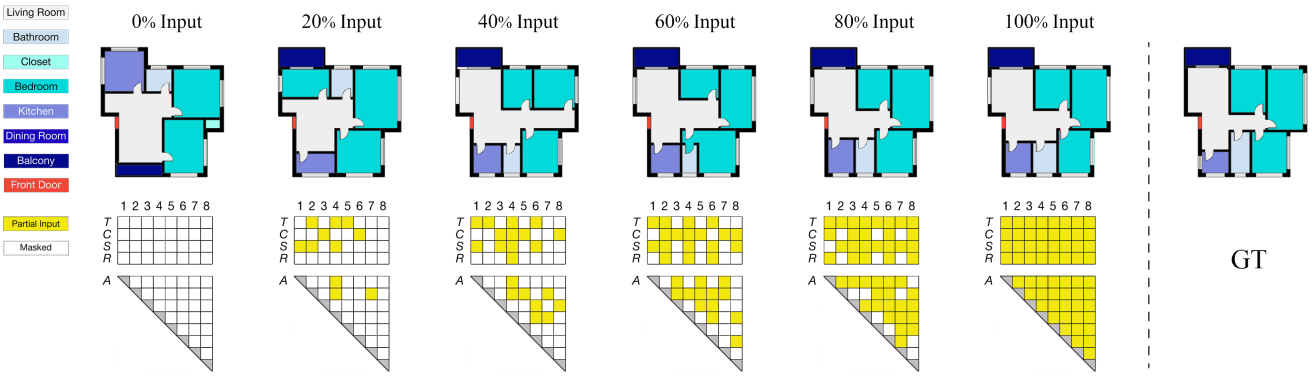

Figure 2. Generated layouts from various ratios of random sampled partial information.

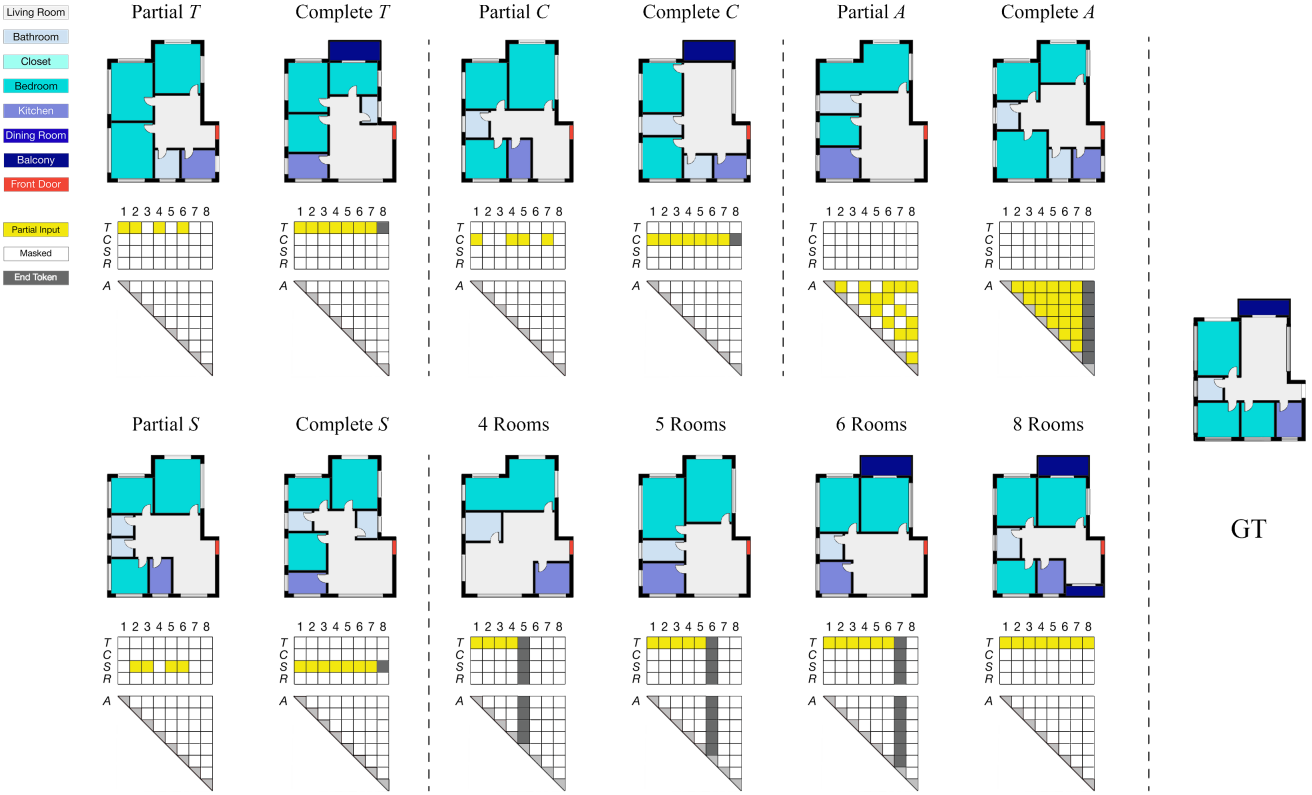

Figure 3. single-attribute guided generations, from partial input or complete input in each feature. MaskPLAN enables new design workflows for designers: (1) predict layouts by simply providing preferred room areas; (2) specify only room adjacency to generate a fitting floorplan; (3) generate feasible layouts with only room locations (even with no room types given).

partial input encoding, MaskPLAN facilitates new design workflows for designers. This is enabled by single-attribute guided generation, allowing users to inform only one of the five generators in the MaskPLAN framework at a time. Synthetic results are shown in Fig. 3. The experiments, utilizing either partial attribute sequences or complete attribute information, demonstrated that complete attribute information significantly enhances layout reconstruction accuracy. Three specific novel workflows include predicting layouts by: (1) providing only a list of preferred room areas (functionally-driven); (2) specifying only room adjacencies (functionally-driven); and (3) a list of locations for unspecified rooms (composition-driven).

References

[1] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 2

[2] Feixiang He, Yanlong Huang, and He Wang. iplan: interactive and procedural layout planning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7793–7802, 2022. 3

[3] Wamiq Para, Paul Guerrero, Tom Kelly, Leonidas J Guibas, and Peter Wonka. Generative layout modeling using constraint graphs. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6690–6700, 2021.

[4] Mohammad Amin Shabani, Sepidehsadat Hosseini, and Yasutaka Furukawa. Housediffusion: Vector floorplan generation via a diffusion model with discrete and continuous denoising. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5466–5475, 2023. 3

[5] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017. 1

[6] Wenming Wu, Xiao-Ming Fu, Rui Tang, Yuhan Wang, Yu-Hao Qi, and Ligang Liu. Data-driven interior plan generation for residential buildings. *ACM Transactions on Graphics (TOG)*, 38(6):1–12, 2019. 3