# Appendix for LayoutDiffusion

## A. More Visualizations

### A.1. Controllable Layout Edit by modifying objects in number, position, size, and category

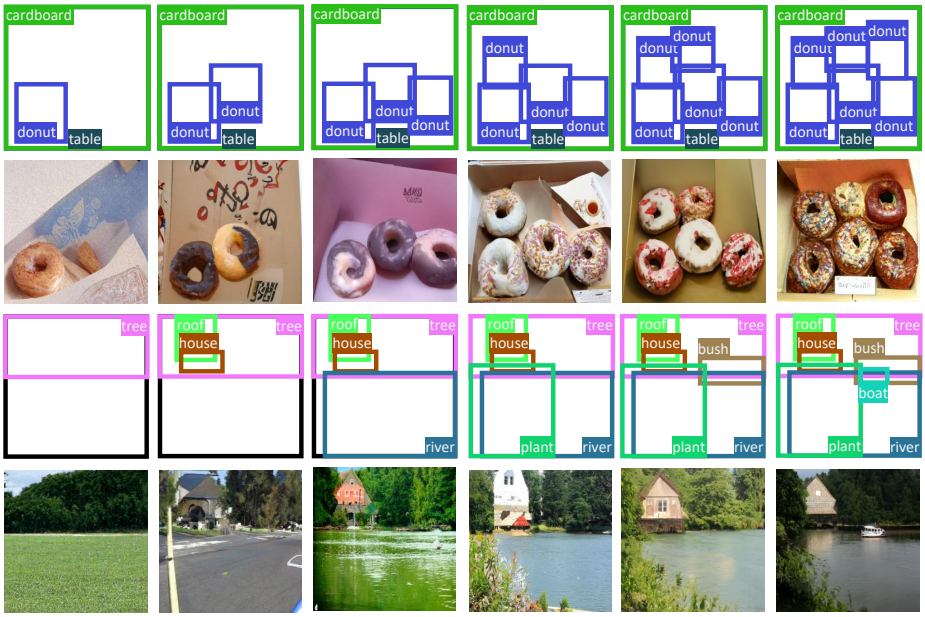


Figure 1. Layout edit by adding objects.

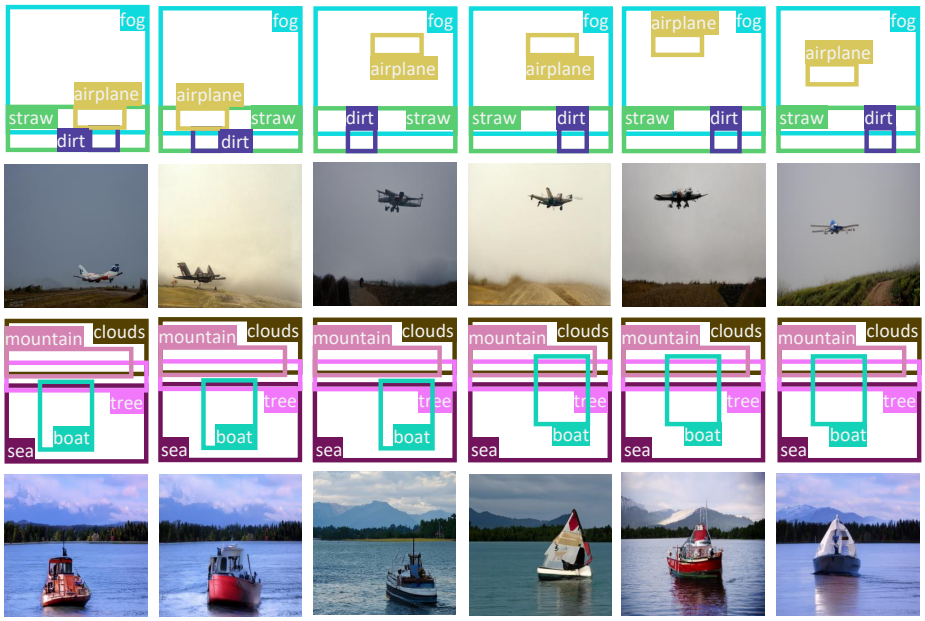


Figure 2. Layout edit by modifying the position of objects.


Figure 3. Layout edit by modifying the size of objects.

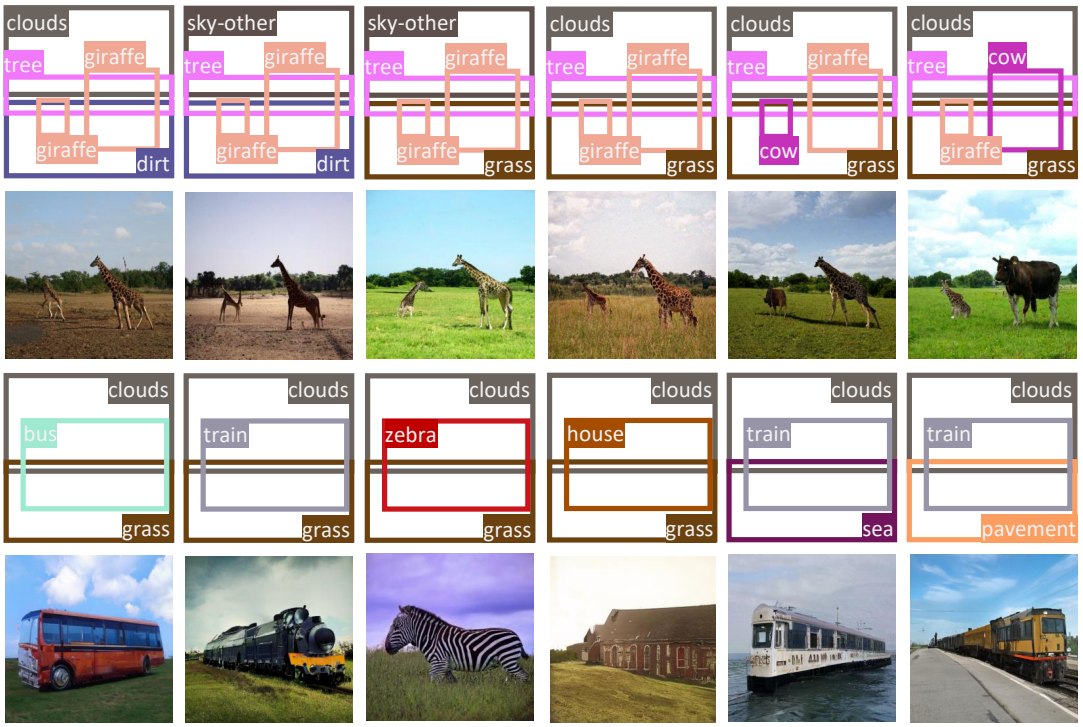


Figure 4. Layout edit by modifying the categories of objects.

## A.2. More visualizations on COCO-stuff

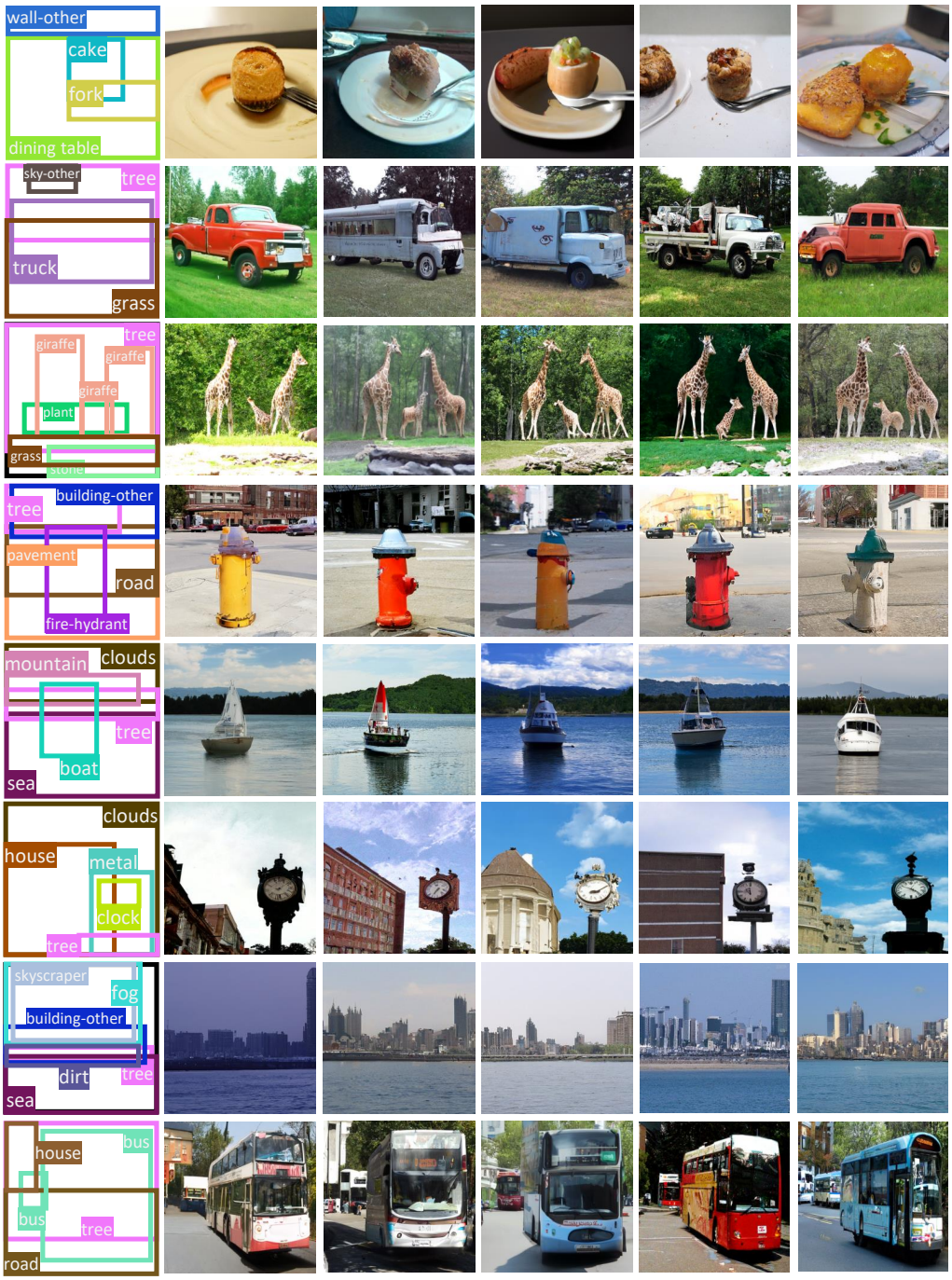


Figure 5. More visualizations on COCO-stuff 256×256. LayoutDiffusion is trained by 1.15M iterations, and sample images using scale=1.0 and dpm-solver 25 steps. The COCO image IDs (from top to bottom) are 85195, 174004, 296969, 338560, 451090, 512248, 573008, 574425.

## A.3. More visualizations on Visual Genome

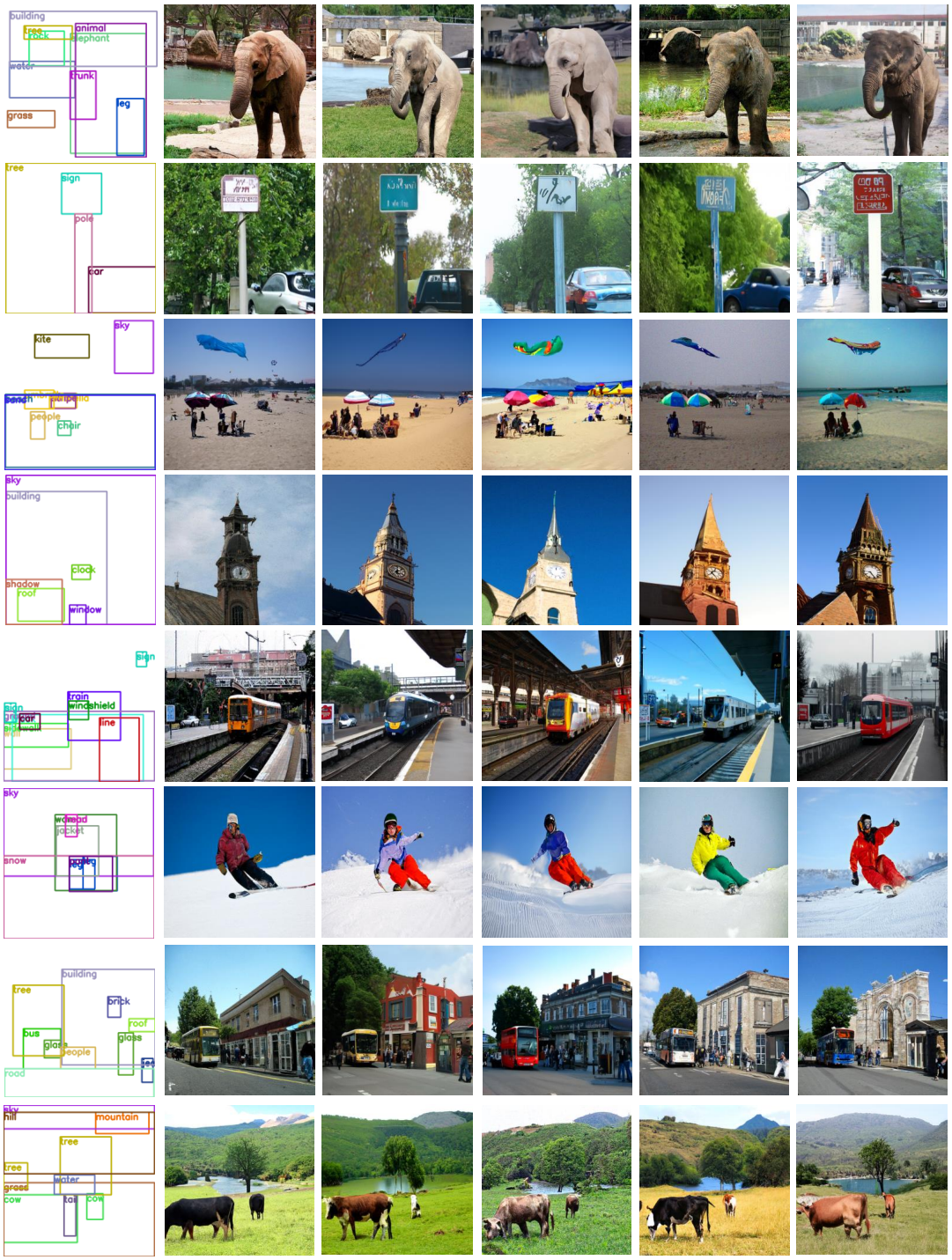


Figure 6. More visualizations on Visual Genome 256×256. LayoutDiffusion is trained by 1.45M iterations, and sample images using scale=1.0 and dpm-solver 25 steps. The VG image IDs (from top to bottom) are 2380568, 2382403, 2382599, 2383021, 2383488, 2385225, 2385290, 2385812.

## A.4. More comparision with previous methods

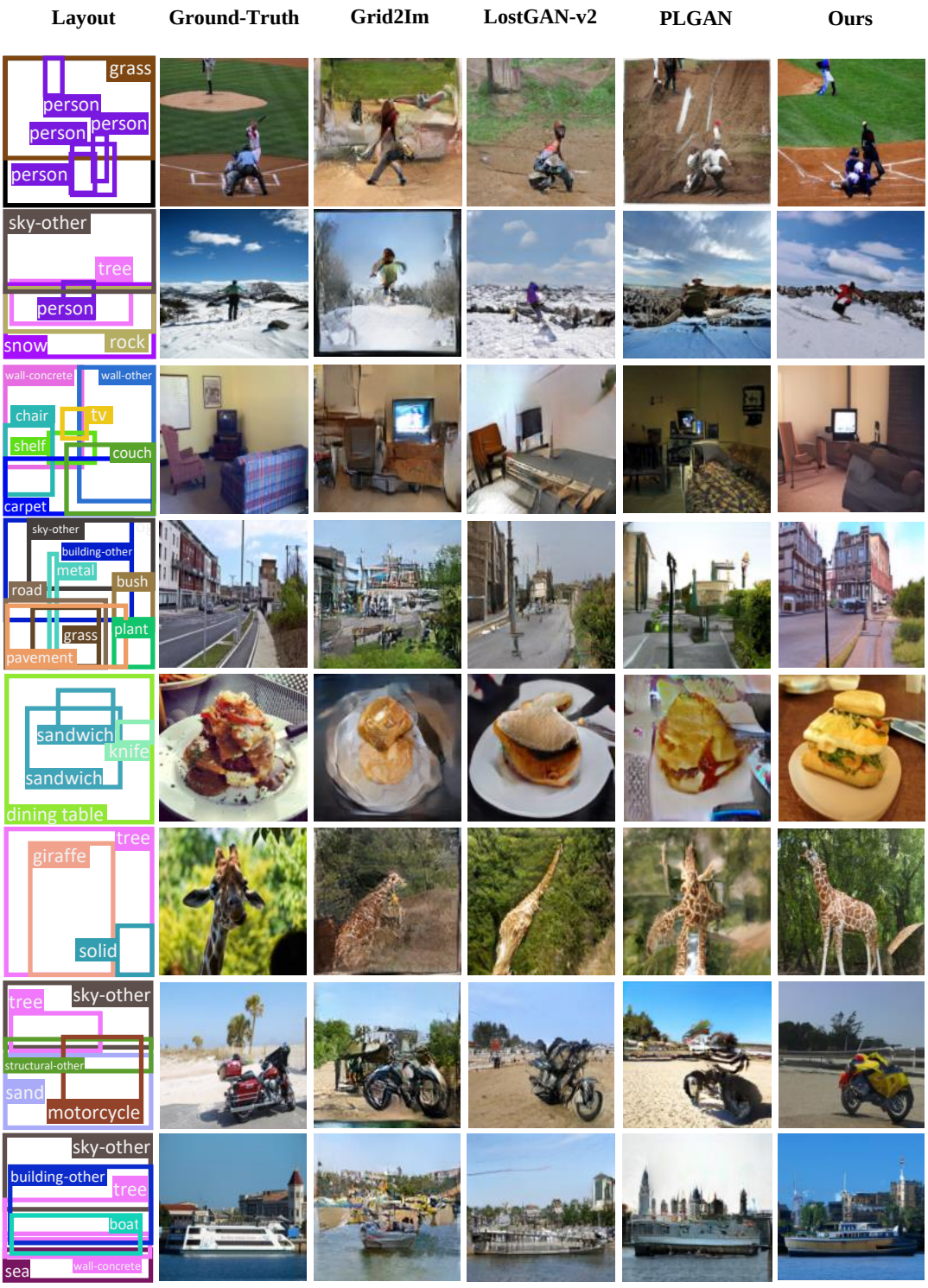


Figure 7. More comparision with previous methods on COCO-stuff 128×128. LayoutDiffusion is trained by 300K iterations, and sample images using scale=0.6 and dpm-solver 25 steps. The COCO image IDs (from top to bottom) are 2153, 2352, 4495, 6723, 10583, 17031, 18737, 543300.

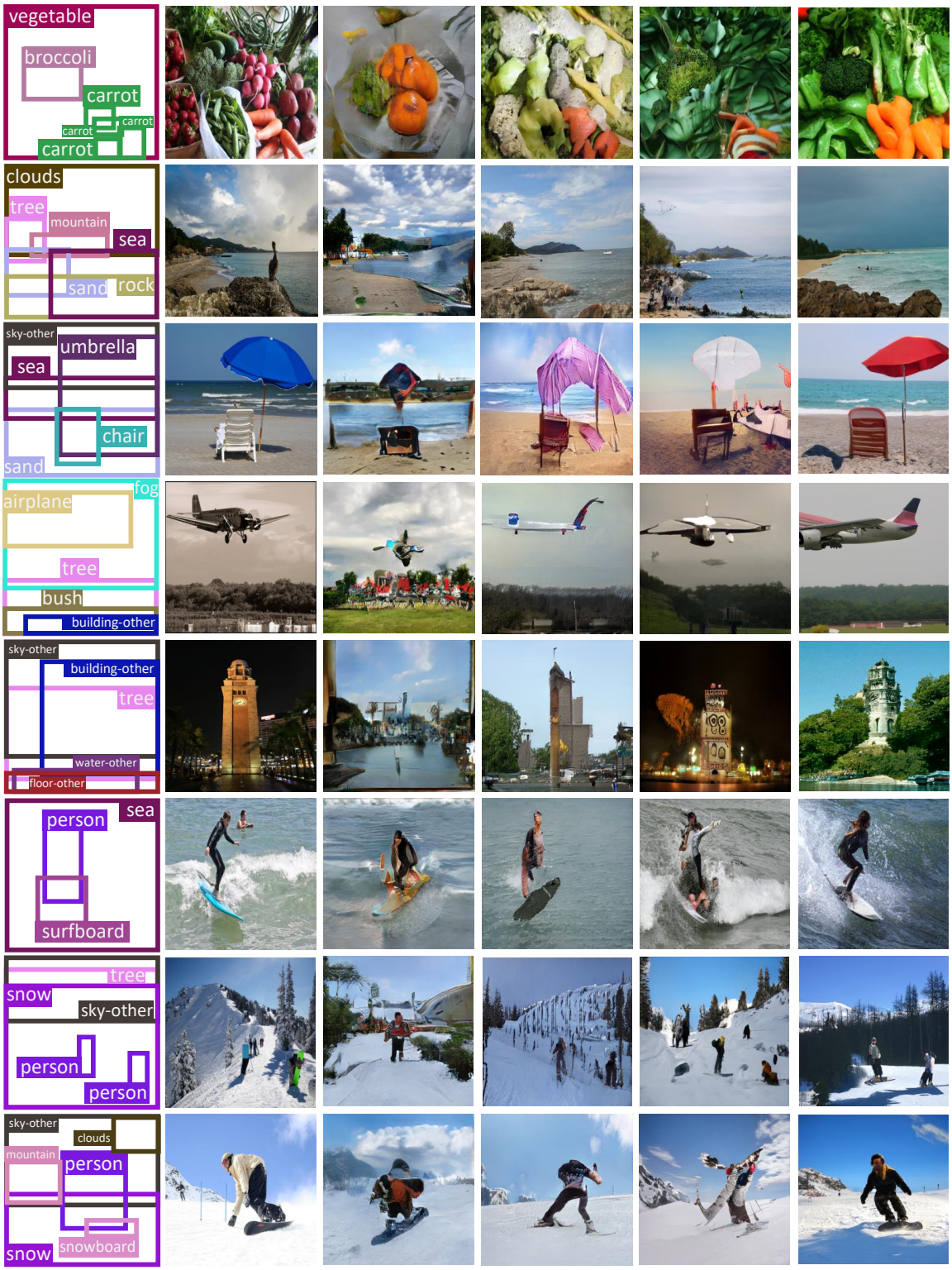


Figure 8. More comparision with previous methods on COCO-stuff 256×256. LayoutDiffusion is trained by 1.15M iterations, and sample images using scale=1.0 and dpm-solver 25 steps. The COCO image IDs (from top to bottom) are 23781, 55299, 84477, 137950, 243034, 252701, 341719, 350405.

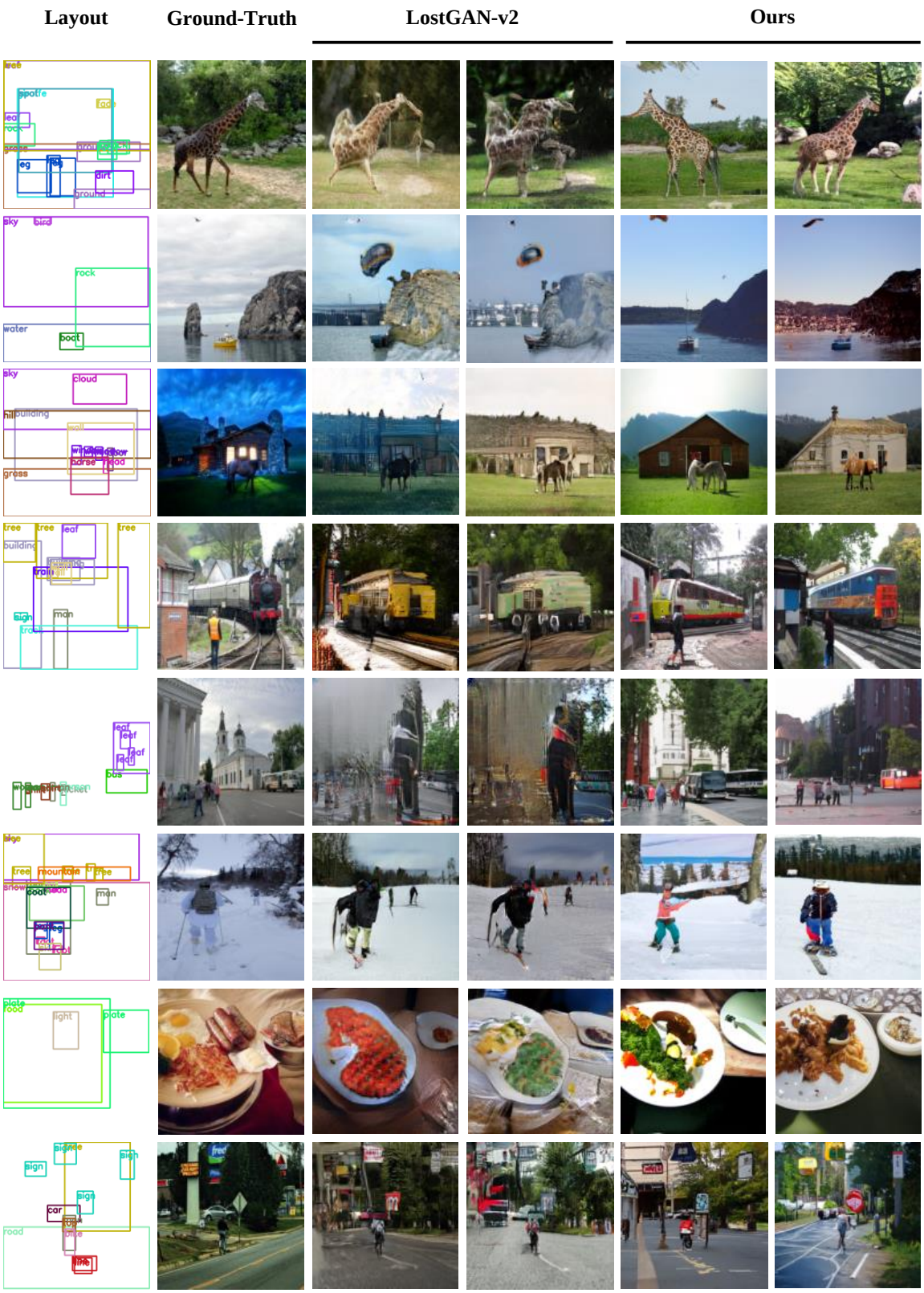


Figure 9. More comparision with previous methods on VG 128×128. LayoutDiffusion is trained by 300K iterations, and sample images using scale=0.5 and dpm-solver 25 steps. The VG image IDs (from top to bottom) are 107945, 150280, 150409, 1160185, 1591817, 1592132, 2341006, 2341475.

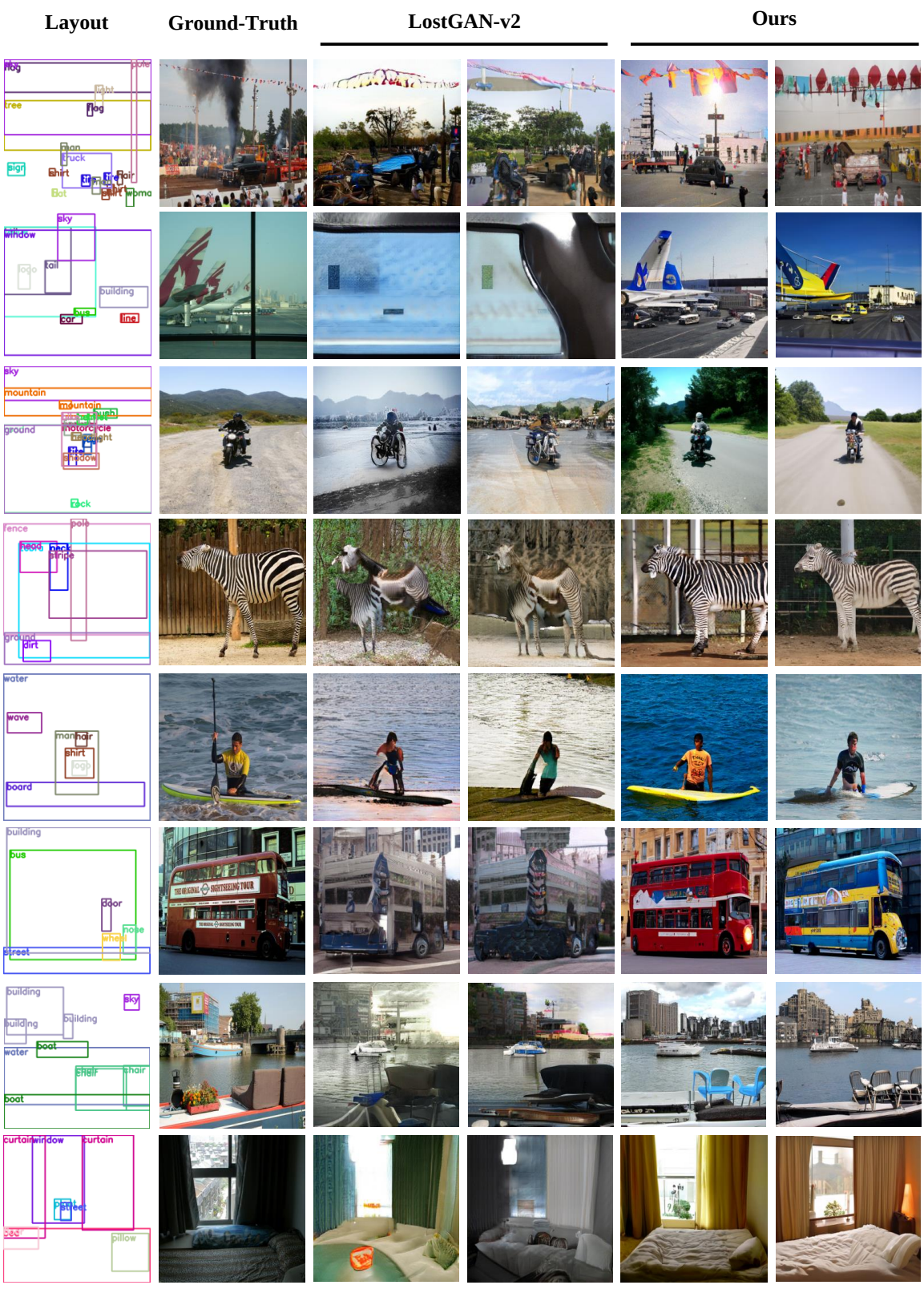


Figure 10. More comparision with previous methods on VG 256×256. LayoutDiffusion is trained by 1.45M iterations, and sample images using scale=1.0 and dpm-solver 25 steps. The VG image IDs (from top to bottom) are 150297, 150358, 1159865, 2340978, 2341300, 2343290, 2344627, 2375966.

### A.5. Different Scales

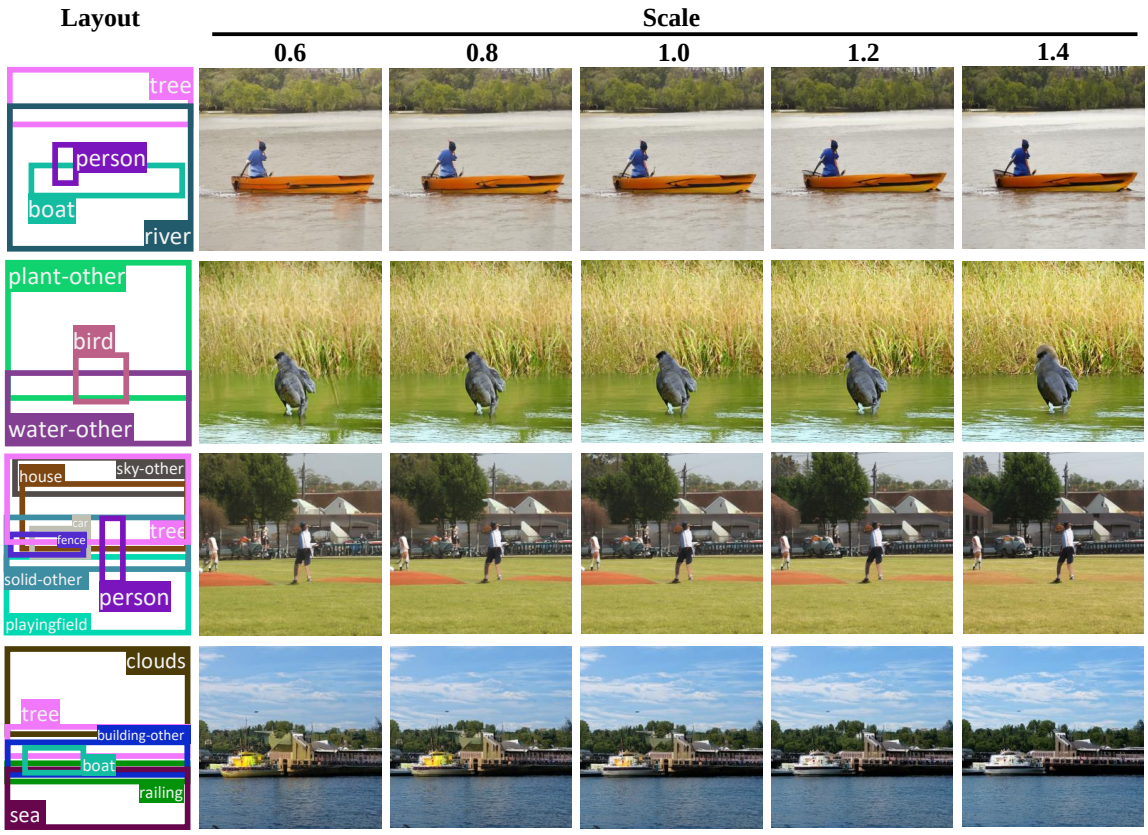


Figure 11. Visualizations on COCO-stuff 256×256 sampled with different scale. LayoutDiffusion is trained by 1.15M iterations, and sample images using dpm-solver 25 steps. The COCO image IDs (from top to bottom) are 472298, 475223, 504415, 513181.

## B. Details of Diffusion Models

### B.1. Denoising Diffusion Probabilistic Model (from ADM-G)

In this section, we will review the formulation of Gaussian diffusion models introduced by DDPM [6]. A data point is defined as $x_0 \sim q(x_0)$. By gradually adding noise to the clean data $x_0$, we can obtain the noised samples from $x_1$ to $x_T$, where $T$ denotes the maximum steps. Specifically, Gaussian noise according to some variance schedule given by $\beta_t$ is added to $x_{t-1}$ in step $t$ of the Markovian noising process $q$:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbf{I}) \tag{1}$$

Due to the convenient nature of Gaussian noise, we do not need to apply $t$ times of $q(x_t|x_{t-1})$ repeatedly to sample from $x_t \sim q(x_t|x_0)$. Instead, $q(x_t|x_0)$ can be directly sampled from a Gaussian distribution:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{2}$$

$$= \sqrt{\bar{\alpha}_t}x_0 + \epsilon\sqrt{1-\bar{\alpha}_t},\ \epsilon \sim \mathcal{N}(0, \mathbf{I}), \tag{3}$$

where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=0}^{t} \alpha_s$. The noise variance for an arbitrary timestep is defined as $1 - \bar{\alpha}_t$ in Eq. (3), and we could equivalently use this to define the noise schedule instead of $\beta_t$. DDPM [6] notes that the posterior $q(x_{t-1}|x_t, x_0)$ is

also a Gaussian using with mean $\tilde{\mu}_t(x_t, x_0)$ and variance $\tilde{\beta}_t$, and is defined as follows:

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu}(x_t, x_0), \tilde{\beta}_t \mathbf{I}) \tag{4}$$

$$\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t} x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} x_t \tag{5}$$

$$\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t \tag{6}$$

If the total noise added throughout the markov chain is large enough when $T \to \infty$ and correspondingly $\beta_t \to 0$, the $x_T$ will be well approximated by $\mathcal{N}(0, \mathbf{I})$. This nice property ensures that we can reverse the above forward process and sample from $x_T \sim \mathcal{N}(0, \mathbf{I})$, which is a Gaussian noise. However, since the entire dataset is needed, we cannot easily estimate the posterior $q(x_{t-1}|x_t, x_0)$. Instead, we have to learn a model $p_\theta(x_{t-1}|x_t)$ to approximate it:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{7}$$

To ensure that the estimated $p_\theta(x_{t-1}|x_t)$ can learrn the true data distribution $q(x_0)$, we can optimize the following variational lower bound $L_{\text{vlb}}$ for $p_\theta(x_0)$:

$$L_{\text{vlb}} = L_0 + L_1 + ... + L_{T-1} + L_T \tag{8}$$

$$L_0 = -\log p_\theta(x_0|x_1) \tag{9}$$

$$L_{t-1} = D_{KL}(q(x_{t-1}|x_t, x_0) \;||\; p_\theta(x_{t-1}|x_t)) \tag{10}$$

$$L_T = D_{KL}(q(x_T|x_0) \;||\; p(x_T)) \tag{11}$$

Although the above objective is well justified, DDPM [6] applied a different objective that produces better samples in practice. Specifically, they do not directly predict $\mu_\theta(x_t, t)$ as the output of a neural network, but instead train a model $\epsilon_\theta(x_t, t)$ to predict $\epsilon$ from Equation 3. This simplified objective is defined as follows:

$$L_{\text{simple}} = E_{t\sim[1,T], x_0\sim q(x_0), \epsilon\sim\mathcal{N}(0,\mathbf{I})}[||\epsilon - \epsilon_\theta(x_t, t)||^2] \tag{12}$$

Then, we can derive $\mu_\theta(x_t, t)$ from $\epsilon_\theta(x_t, t)$ using the following substitution:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right) \tag{13}$$

Note that $\Sigma_\theta(x_t, t)$ is not learned in $L_{\text{simple}}$ in DDPM [6] and is fixed as a constant such as $\beta_t\mathbf{I}$ or $\tilde{\beta}_t\mathbf{I}$, corresponding to upper and lower bounds for the true reverse step variance [16], respectively.

### B.2. Classifier-free Method for Layout-conditional Training and Sampling

Instead of training a separate classifier model, Ho & Salimans [7] choose to train a diffusion model that allows for both conditional and unconditional sampling, where the unconditional diffusion model $p_\theta(x_t, t)$ is parameterized through a score estimator $\epsilon_\theta(x_t, t)$ and the conditional diffusion model $p_\theta(x_t, t|c)$ is parameterized through $\epsilon_\theta(x_t, t, c)$.

They use a single model to parameterize both conditional and unconditional models, where for the unconditional model they simply input a null token $\varnothing$ for the class identifier $c$ when predicting the score, i.e. $\epsilon_\theta(x_t, t) = \epsilon_\theta(x_t, t, c = \varnothing)$. They jointly train this unified model by randomly replacing $c$ with the unconditional class identifier $\varnothing$ with probability $p_{\text{uncond}}$. Then the conditional score estimate $\epsilon_\theta(x_t, t, c)$ is replaced with $\tilde{\epsilon}_\theta(x_t, t, c)$ using the following equation:

$$\tilde{\epsilon}_\theta(x_t, t, c) = \epsilon_\theta(x_t, t, c) + s(\epsilon_\theta(x_t, t, c) - \epsilon_\theta(x_t, t)), \tag{14}$$

which can be considered as a linear combination of conditional and unconditional score estimates. $s$ is the scale and Since Eq. (14) has no classifier gradient, no more gradient calculation in classifier guidance is needed during sampling. Furthermore, $s$ can be changed to modify the effect of the condition.

In the layout-to-image generation, $c$ is the layout $l$ defined in Sec. 3.1. Layout Embedding and $\varnothing$ is the emtpy layout $l_{\text{pad}}$ defined in Sec. 3.4. Layout-conditional Diffusion Model.

# C. Implementation details

## C.1. Hyperparameters

| **Dataset** | COCO-stuff 256×256 | | COCO-stuff 128×128 | VG 256×256 | VG 128×128 |
|---|---|---|---|---|---|
| **Model** | LayoutDiffusion | LayoutDiffusion-small | LayoutDiffusion | LayoutDiffusion | LayoutDiffusion |
| Layout-conditional Diffusion Model | | | | | |
| In Channels | 3 | 3 | 3 | 3 | 3 |
| Out Channels | 6 | 6 | 6 | 6 | 6 |
| Hidden Channels | 256 | 128 | 256 | 256 | 256 |
| Channel Multiply | 1,1,2,2,4,4 | 1,1,2,2,4,4 | 1,1,2,3,4 | 1,1,2,2,4,4 | 1,1,2,3,4 |
| Number of Residual Blocks | 2 | 2 | 2 | 2 | 2 |
| Dropout | 0 | 0 | 0.1 | 0.1 | 0.1 |
| Diffusion Steps | 1000 | 1000 | 1000 | 1000 | 1000 |
| Noise Schedule | linear | linear | linear | linear | linear |
| Layout-Image Fusion Module | | | | | |
| Downsampling Scale For Fusion | 8,16,32 | 8,16,32 | 4,8,16 | 8,16,32 | 4,8,16 |
| Resolution for Fusion | 32,16,8 | 32,16,8 | 32,16,8 | 32,16,8 | 32,16,8 |
| Fusion Method | OaCA | OaCA | OaCA | OaCA | OaCA |
| Number of Attention Blocks | 1 | 1 | 1 | 1 | 1 |
| Number of Heads | 4 | 4 | 4 | 4 | 4 |
| Layout Fusion Module | | | | | |
| Hidden Channels | 256 | 128 | 256 | 256 | 256 |
| Transformer Depth | 6 | 4 | 6 | 6 | 6 |
| Attention Method | Self-Attention | Self-Attention | Self-Attention | Self-Attention | Self-Attention |
| Number of Heads | 8 | 8 | 8 | 8 | 8 |
| Layout Embedding | | | | | |
| Embedding Dimension | 256 | 128 | 256 | 256 | 256 |
| Maximum Number of Objects | 8 | 8 | 8 | 10 | 10 |
| Maximum Number of Length | 10 | 10 | 10 | 12 | 12 |
| Maximum Number of Class Id | 185 | 185 | 185 | 180 | 180 |
| Training Hyperparameters | | | | | |
| Total Batch Size | 32 | 32 | 64 | 32 | 64 |
| Number of GPUs | 8 | 8 | 8 | 8 | 8 |
| Learning Rate | 1e-5 | 1e-5 | 2e-5 | 1e-5 | 2e-5 |
| Mixed Precision Training | Yes | Yes | Yes | Yes | Yes |
| Weight Decay | 0 | 0 | 0 | 0 | 0 |
| EMA Rate | 0.9999 | 0.9999 | 0.9999 | 0.9999 | 0.9999 |
| Classifier-free Dropout | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| Iterations | 1.15M | 1.4M | 300K | 1.45M | 300K |

Table 1. Hyperparameters for the proposed LayoutDiffusion in Sec. 4.4. Quantitative results. All trained on eight RTX 3090.

| **Model** | LayoutDiffusion | − LFM | − OaCA | −OaCA, + CA | − LFM, − OaCA |
|---|---|---|---|---|---|
| Fusion Method | OaCA | OaCA | - | CA | - |
| Transformer Depth | 6 | 0 | 6 | 6 | 0 |
| Iterations | 300K | 300K | 300K | 300K | 300K |
| FID | 16.57 | + 0.19 | + 0.49 | - 0.11 | + 13.37 |
| DS | 0.47 | + 0.01 | + 0.05 | + 0.01 | + 0.23 |
| CAS | 43.60 | - 2.93 | - 12.74 | - 1.13 | - 39.77 |
| YOLOScore | 27.00 | - 8.20 | - 20.10 | - 3.4 | - 27.0 |

Table 2. Hyperparameters for the ablation study in Sec. 4.5 .

### C.2. Analysis of Training Resources and Sampling Speed

| method | FID ↓ | $N_{\text{parms}}$ | Throughout images / s | Compression stage V100 days | Diffusion stage V100 days | Total V100 days |
|---|---|---|---|---|---|---|
| LDM-8 (100 steps) | 42.06 | 345M | 0.457 | 66 | 3.69 | 69.69 |
| LDM-4 (200 steps) | 40.91 | 306M | 0.267 | 29 | 95.49 | 124.49 |
| LayoutDiffusion-small (25 steps) | 36.16 | 142M | 0.608 | - | 75.83 | 75.83 |
| LayoutDiffusion (25 steps) | 31.68 | 569M | 0.308 | - | 216.55 | 216.55 |

Table 3. Comparison with SOTA diffusion-based methods LDM on COCO-stuff 256×256. We generate the same 2048 images of LDM for a fair comparision. LDM is sampled using DDIM [16] and LayoutDiffusion is sampled with DPM-Solver [12]. The hyperparameters of LayoutDiffusion and LayoutDiffusion-small are listed in Tab. 1

## D. Evaluation

### D.1. Datasets

**COCO-Stuff** [3]. COCO [11] is a large-scale object detection, segmentation, and captioning dataset. COCO-Stuff [3] augments all 164K images of the COCO [11] dataset with pixel-level stuff annotations. These annotations can be used for scene understanding tasks like semantic segmentation, object detection and image captioning. COCO-Stuff [3] contains 80 categories of thing and 91 categories of stuff, respectively. Following the settings of LostGAN-v2 [17], we use the COCO 2017 Stuff Segmentation Challenge subset containing 40K / 5k / 5k images for train / val / test-dev set. Segmengtation annotation is not used. We use images in the train and val set with 3 to 8 objects that cover more than 2% of the image and not belong to 'crowd'. Finally, there are 25,210 train and 3,097 val images. Some previous works [1, 8] don't filter objects belong to 'crowd', causing different numbers of images. Specificly, there are 24,972 train and 3,074 val images. LAMA [10] uses the full COCO-Stuff [3] 2017 dataset, leaving 74,777 train and 3,097 val images. Tab. 4 summarizes the difference in the number of images by different filtering methods.

| **use deprecated** | **filter 'crowd'** | train | | | val | | |
|---|---|---|---|---|---|---|---|
| | | image | object | object/image | image | object | object/image |
| | | 74,121 | 411,682 | 5.55 | 3,074 | 17,100 | 5.56 |
| ✓ | | 24,972 | 138,162 | 5.53 | 3,074 | 17,100 | 5.56 |
| | ✓ | 74,777 | 414,443 | 5.54 | 3,097 | 17,191 | 5.55 |
| ✓ | ✓ | 25,210 | 139,175 | 5.52 | 3,097 | 17,191 | 5.55 |

Table 4. The difference in the number of images by different filtering methods, **use deprecated** means use COCO 2017 Stuff Segmentation Challenge subset, **filter 'crowd'** means filter objects belong to 'crowd'.

**Visual Genome** [9]. Following the settings of Sg2Im [8], we experiment on Visual Genome [9] version 1.4 (VG) which comprises 108,077 images annotated with scene graphs. Visual Genome [9] collects images with dense annotations of objects, attributes, and relationships. Here, we only use bounding boxes. We divide the data into 80% / 10% / 10% for the train / val / test set. We select the object / relationship categories occurring at least 2000 / 500 times in the train set, respectively, and select the images with 3 to 30 bounding boxes and ignoring all small objects. Finally, the train / val / test set has 62,565 / 5,062 / 5,096 images.

### D.2. Evaluation Metrics

Comprehensive evaluations of generated images remains a challenge. We use six metrics, from image-level to layout-level, to evaluate the quality of the generated images and the layout control from different aspects.

**Fr'echet Inception Distance (FID)** [5] shows the difference between the real images and the generated images by using an ImageNet-pretrained Inception-V3 [18] network and computing the Fr'echet distance between two Gaussian distributions

fitted to generated images and real images respectively. We save the GT images as real images when sampling the generated images. The real images and the generated images are saved to two folder respectively. Then compute the FID score, based on the official code of FID[1]. For FID, the lower the score, the smaller the difference between the generated images and the real images, meaning the generator model is better.

**Inception Score (IS)** [15] shows the overall quality of the generated images by using an Inception-V3 [18] network pretrained on the ImageNet-1000 classification benchmark and computing a score(statistics) of the network's outputs with generated images of a generator model. IS measures the quality of images on two aspects: clarity and diversity. We only need the generated images to compute the IS score, based on the official code of IS[2]. For IS, the higher the score, the better the quality of generated images, meaning the generator model is better.

**Diversity Score (DS)** measures the diversity between the generated images from the same layout by comparing the perceptual similarity in a DNN feature space between them. Here, we adopt the LPIPS [19] metric. For each sample, we repeat two times, use the two images from the same layout to compute the DS score, then calculate mean and std of these scores as the reported DS score, based on the official code of DS[3]. For DS, the higher the score, the better the diversity between the generated images, meaning the generator model is better.

**YOLO Score** [10] uses a pretrained YOLOv4 [2] model to evaluate bbox mAP on 80 thing categories based on the official code of LAMA[4] and the official code of YOLOv4[5]. YOLO Score [10] is proposed to evaluate the alignmentand fidelity of generated objects, measuring how generated objects are recognizable when even the layout is unknow. YOLO [2, 14] is a well-known series of object detector, inferring layouts from the given images. Before send images to detector, they are upsampled to 512×512. And different from LAMA [10], since we filter the objects and images in datasets, we think it is better to evaluate bbox mAP only on filtered annotations.

**Classification Score (CAS)** [13] measures classification accuracy of layout areas on generated images. We crop the GT box area of images and resize objects at a resolution of 32×32 with their class. Then train a ResNet101 [4] classifier with cropped images on generated images and test it on cropped images on real images, based on a widely used codebase of image classification[6]. For CAS, the higher the score, the better the quality of layout control, meaning the generator model is better.

[1] the official code of FID [5]:https://github.com/bioinf-jku/TTUR
[2] the official code of IS [15]:https://github.com/openai/improved-gan
[3] the official code of DS [19]:https://github.com/richzhang/PerceptualSimilarity
[4] the official code of LAMA [10]:https://github.com/ZejianLi/LAMA
[5] the official code of YOLOv4 [2]:https://github.com/AlexeyAB/darknet
[6] the widely used codebase of image classification: https://github.com/hysts/pytorch_image_classification

## References

[1] Oron Ashual and Lior Wolf. Specifying object attributes and relations in interactive scene generation. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019. 12

[2] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020. 13

[3] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. Coco-stuff: Thing and stuff classes in context. In *CVPR*, pages 1209–1218, 2018. 12

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 13

[5] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, pages 6626–6637, 2017. 12, 13

[6] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020. 9, 10

[7] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021. 10

[8] Justin Johnson, Agrim Gupta, and Li Fei-Fei. Image generation from scene graphs. In *CVPR*, pages 1219–1228, 2018. 12

[9] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, et al. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *IJCV*, 123(1):32–73, 2017. 12

[10] Zejian Li, Jingyu Wu, Immanuel Koh, Yongchuan Tang, and Lingyun Sun. Image synthesis from layout with locality-aware mask adaption. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13819–13828, 2021. 12, 13

[11] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014. 12

[12] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *arXiv preprint arXiv:2206.00927*, 2022. 12

[13] Suman Ravuri and Oriol Vinyals. Classification accuracy score for conditional generative models. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. 13

[14] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016. 13

[15] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *NIPS*, 2016. 13

[16] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv:2010.02502*, October 2020. 10, 12

[17] Wei Sun and Tianfu Wu. Learning layout and style reconfigurable gans for controllable image synthesis. *IEEE TPAMI*, 44(9):5070–5087, 2021. 12

[18] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015. 12, 13

[19] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 13