MIGC: Multi-Instance Generation Controller for Text-to-Image Synthesis

Supplementary Material

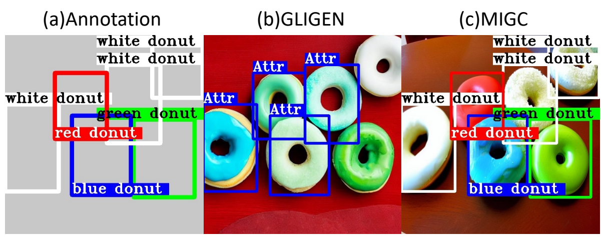

Figure 1. An example from the COCO-MIG benchmark. (a) In this example, COCO-MIG requires generation models to generate “donuts” of various colors according to the specified positions and color attributes. (b) Although the state-of-the-art layout-to-image method GLIGEN can generate “donuts” according to the specified position in this example, their color attributes are not correct. We use boxes with “Attr” to mark the wrong color attributes. (c) Our proposed MIGC can not only generate “donuts” according to the position specified by the annotation but also ensure that the color attribute of each generated donut instance is correct.

A. Construction Process of COCO-MIG Benchmark

Overview. COCO-MIG benchmark uses the layout of COCO-position benchmark [13] and assigns a specific **color** attribute to each instance. COCO-MIG requires that each instance generated not only meet the position requirements but also meet the attribute (i.e., color) requirements.

Step 1: Sampling layouts from COCO. We sample layouts from the COCO-position [13], filter out instances with side lengths less than 1/8 of the original image size, and further filter out those layouts with less than two instances. To test the model’s ability to control quantity, we divided these layouts into five levels, L_2-L_6, based on the number of instances, where L_i indicates that there are i instances in the target-generated image. A total of 160 layouts are sampled for each level. Notably, in the process of sampling layouts for level L_i, if the number of instances surpasses i, we selectively choose the initial i instances with the largest area. Conversely, if the number of instances is less than i, a resampling procedure is employed.

Step 2: Assigning color attribute to each instance. On the basis of each sampled layout, we assign each instance a specific color from eight colors, i.e., red, yellow, green, blue, white, black, and brown. At the same time, we write the global prompt as ’a <attr1> <obj1>, a <attr2> <obj2>, ..., and a ...’.

B. Difference between COCO-MIG and COCO-position

Fig. 1 shows a specific example. In this example, COCO-MIG assigns a specific color to each “donut” based on the COCO-position layout. Fig. 1(b) shows the results of the state-of-the-art layout-to-image method GLIGEN [12]. It can be seen that the results generated by GLIGEN meet the position requirements, so this will be judged as correctly generated in the COCO-positon benchmark. However, COCO-MIG not only requires the generated instances to meet position requirements but also attributes requirements. From this perspective, COCO-MIG will determine that the results generated by GLIGEN are incorrectly generated because there are “donuts” of incorrectly generated color attributes. Finally, it can be seen that using our proposed MIGC guarantees that the position and attributes of each generated instance are correct.

C. More MIG Results

Fig. 2 and Fig. 3 show more results obtained using MIGC for Multi-Instance Generation. Even with complex layouts and rich attribute descriptions, MIGC can ensure that each instance is generated at the correct position and has the correct attributes. At the same time, if the relationship between each instance is specified in the global prompt (e.g., action relationship), MIGC can further control the interaction between instances.

D. More Qualitative Results on COCO-MIG

More qualitative results on our proposed COCO-MIG benchmark are shown in Fig. 4. Compared with previous state-of-the-art methods, our proposed MIGC approach can better control the position, attributes, and quantity simultaneously.

E. Qualitative Results on DrawBench

Implementation details. DrawBench [18] is a challenging T2I benchmark. On this benchmark, we compare our proposed MIGC with state-of-the-art text-to-image (i.e., AAE [2], Struc-D [4]) and layout-to-image methods (i.e., Box-D [20], TFLCG [3], Multi-D [1], GLIGEN [12]). For the text-to-image methods, we directly input the DrawBench’s prompt into the pipeline. For the layout-to-image methods and our proposed MIGC, we first use GPT-4 [5, 16] to generate the layout and then input it into the network, forming a two-stage text-to-image pipeline.

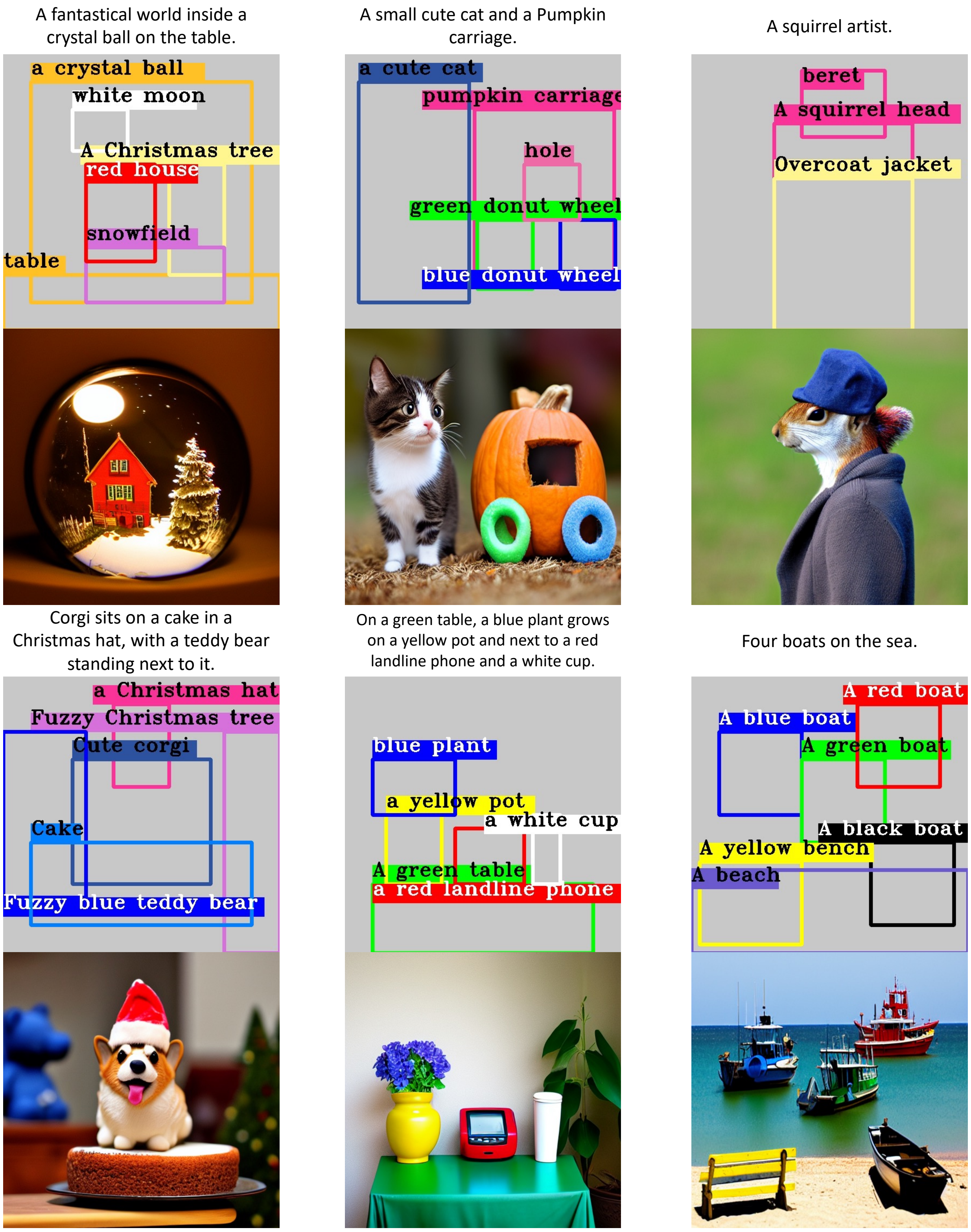

Figure 2. Multi-Instance Generation (MIG) with our MIGC. MIGC can generate images based on various complex layouts and ensure that the attributes of each instance are correct.

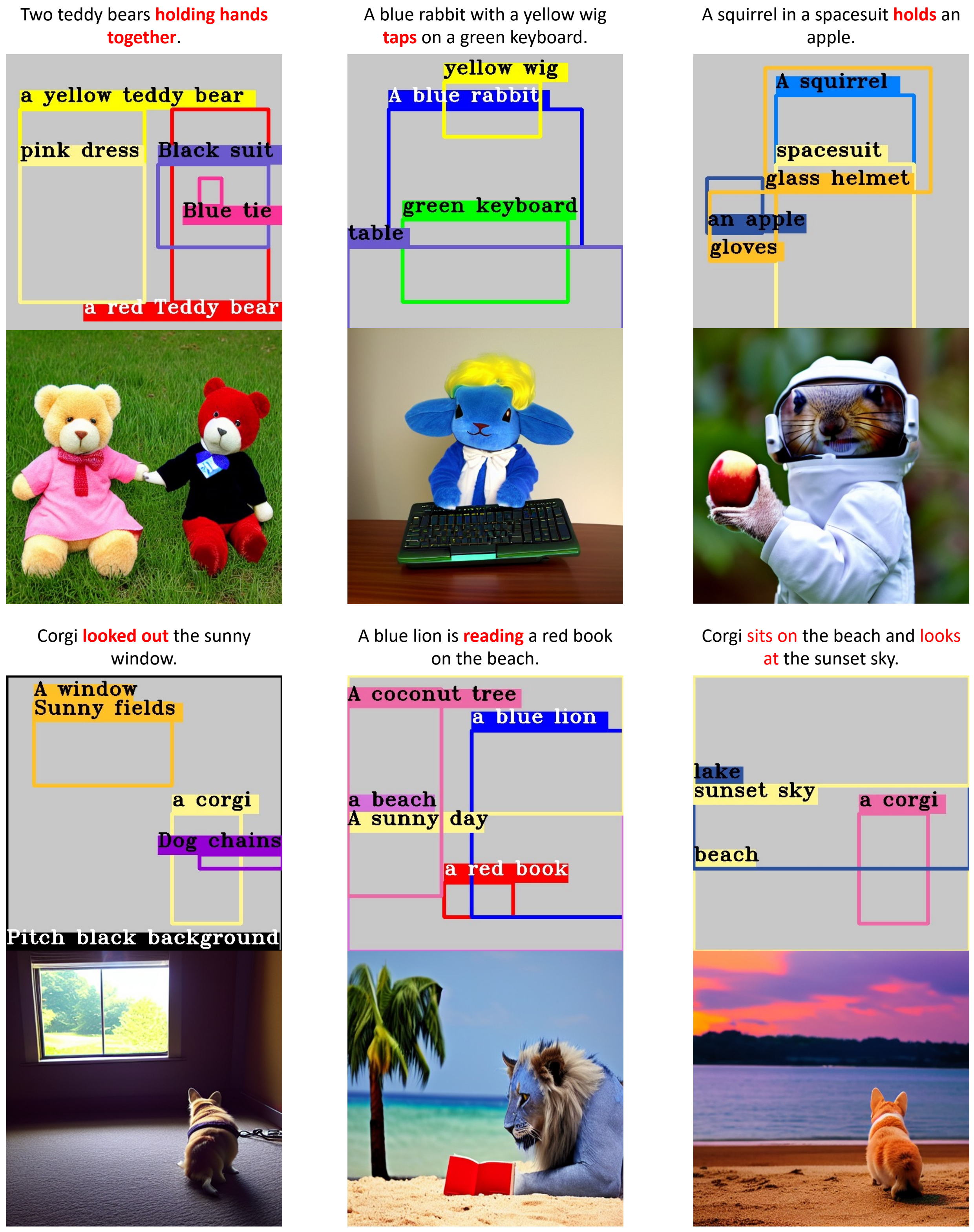

Figure 3. Multi-Instance Generation (MIG) with our MIGC. By specifying the relation between instances through the global prompt, MIGC can further control the interaction of instances.

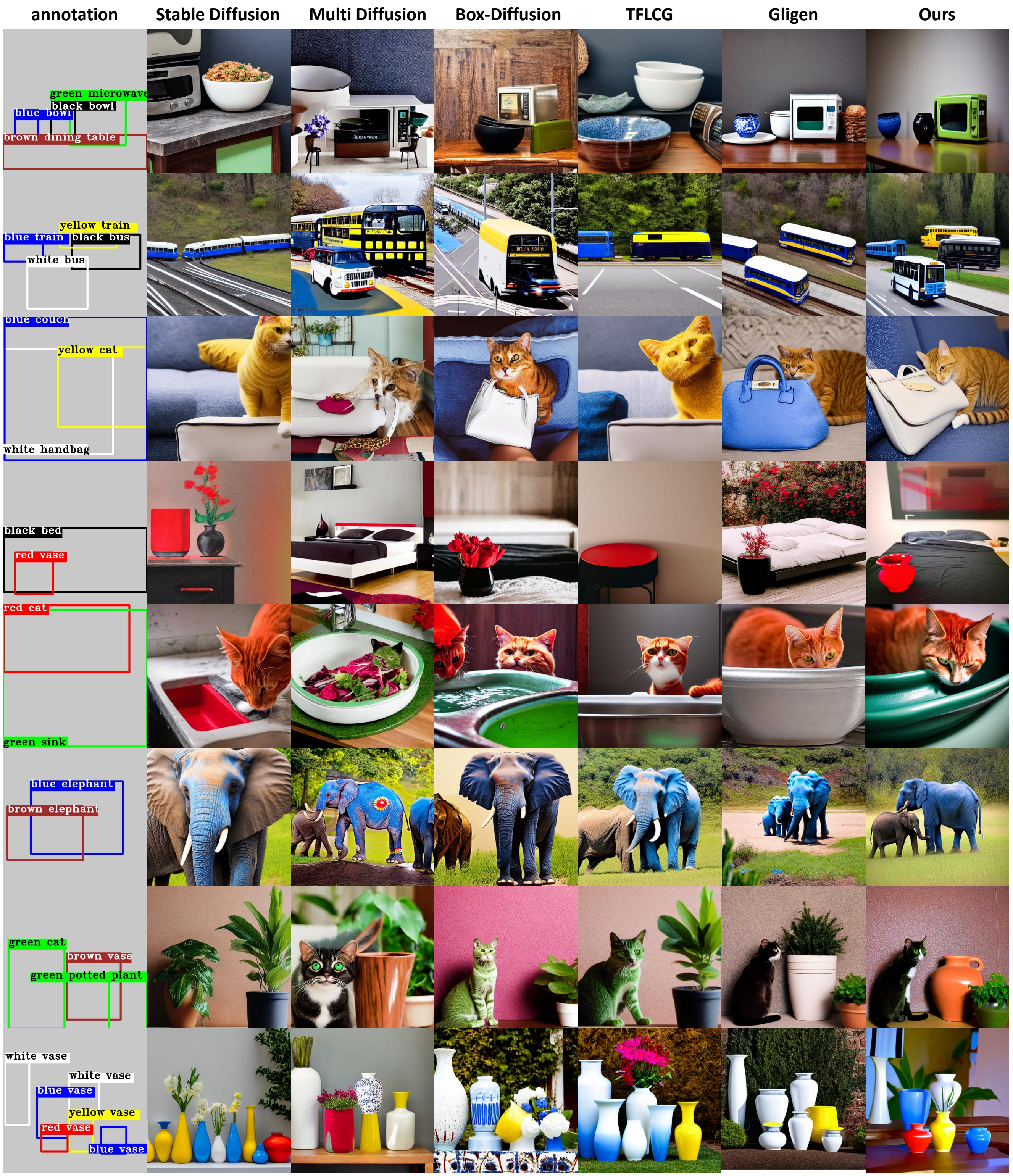

Figure 4. More qualitative results of the proposed COCO-MIG benchmark. The first column on the left shows the target layout and descriptions for each instance, and we use the corresponding colored boxes to describe the target color for each instance. The columns on the right display the results of baseline and MIGC, respectively. To better observe the details of the image, we did not add any extra annotations to the image.

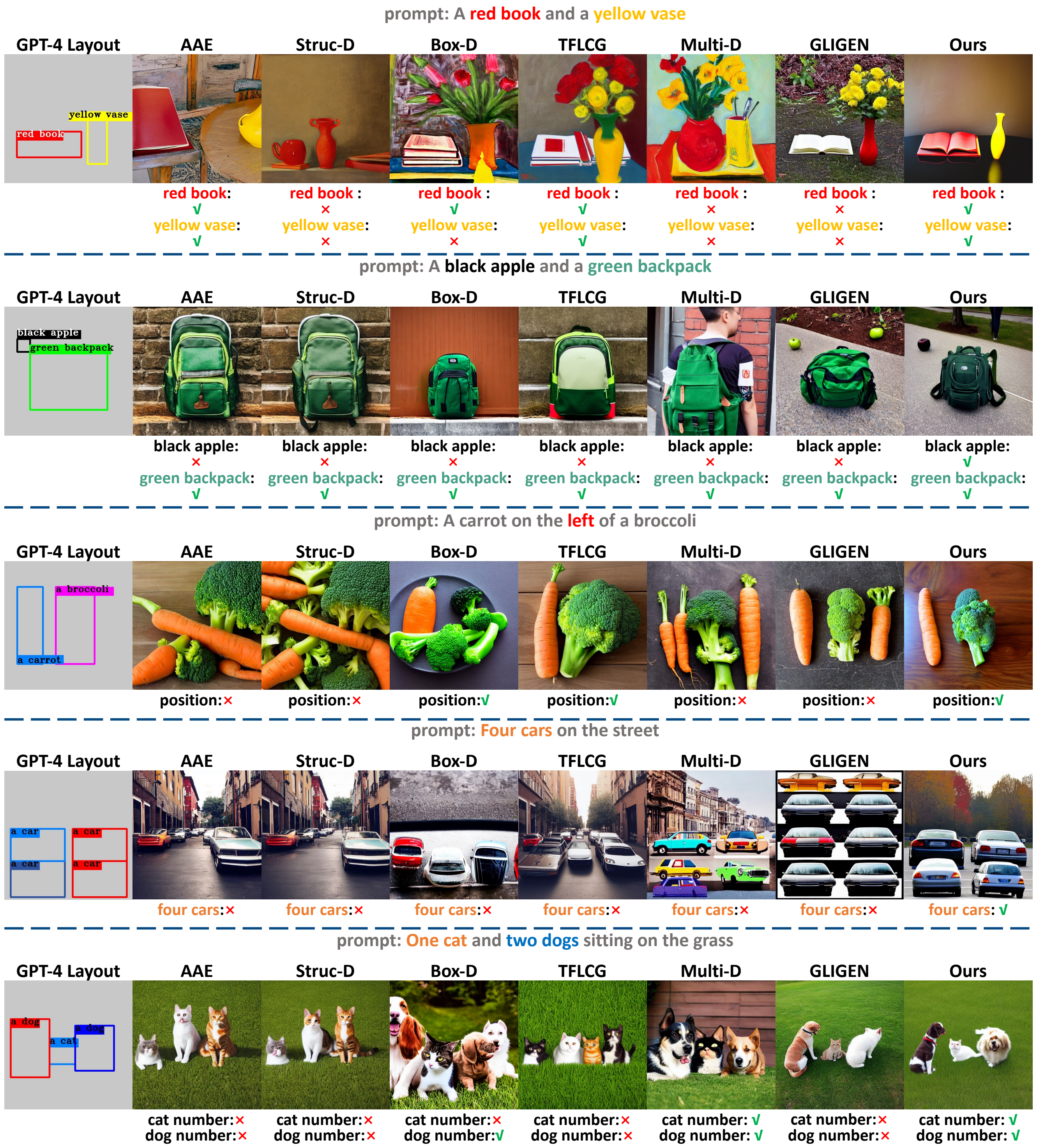
prompt: A red book and a yellow vase
GPT-4 Layout
AAE
Struc-D
Box-D
TFLCG
Multi-D
GLIGEN
Ours
yellow vase
red book
red book: √ yellow vase: √
red book : × yellow vase: ×
red book : √ yellow vase: ×
red book : √ yellow vase: √
red book : × yellow vase: ×
red book : × yellow vase: ×
red book : √ yellow vase: √
prompt: A black apple and a green backpack
black apple
green backpack
black apple: × green backpack: √
black apple: × green backpack: √
black apple: × green backpack: √
black apple: × green backpack: √
black apple: × green backpack: √
black apple: × green backpack: √
black apple: √ green backpack: √
prompt: A carrot on the left of a broccoli
a broccoli
a carrot
position:×
position:×
position:√
position:√
position:×
position:×
position:√
prompt: Four cars on the street
a car
a car
a car
a car
four cars:×
four cars:×
four cars:×
four cars:×
four cars:×
four cars:×
four cars: √
prompt: One cat and two dogs sitting on the grass
a dog
a cat
a dog
cat number:× dog number:×
cat number:× dog number:×
cat number:× dog number:√
cat number:× dog number:×
cat number: √ dog number: √
cat number:× dog number:×
cat number: √ dog number: √

Figure 5. Qualitative comparison of DrawBench. The first and second rows show the results in the "COLOR" subtask, in which MIGC achieves precise attribute control while obvious attribute leakage problems appear in other state-of-the-art methods. The third row shows that MIGC can achieve precise position control. The fourth and fifth rows show that MIGC can achieve precise quantity control, especially in the case where "cat" and "dog" exist at the same time, in which MIGC avoids the mutual influence of cat and dog semantics since MIGC can achieve good attribute control.

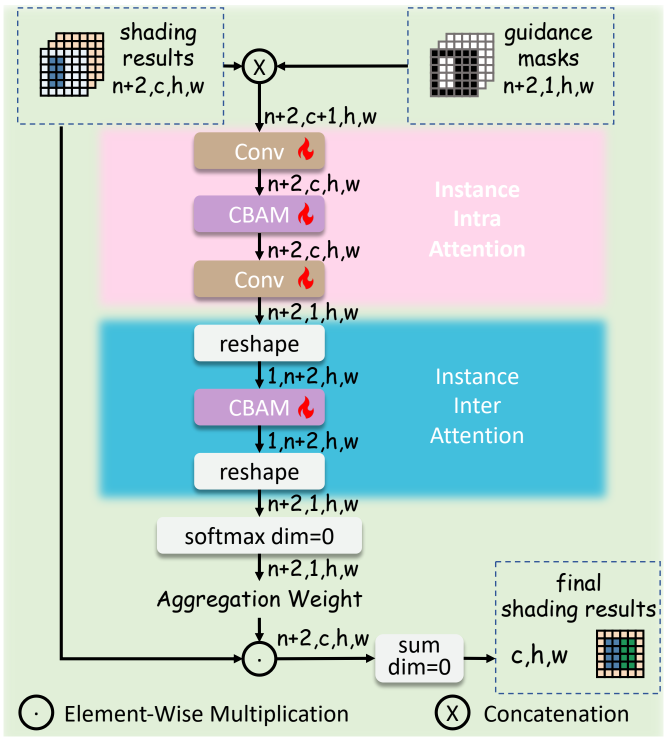

Figure 6. Architecture details of Shading Aggregation Controller.

Qualitative results. Fig. 5 shows the qualitative comparison on DrawBench. The first row shows that obvious attribute leakage problems (i.e., confusion between the "yellow" and "red" attributes) occur in the results of previous state-of-the-art methods, while our MIGC can control the attributes of each instance very precisely. The second row shows that previous state-of-the-art methods cannot correctly generate a "black apple," which is counterfactual, while our MIGC can achieve good generation. The third row indicates that MIGC can control the position more accurately than the previous state-of-the-art method and can effectively solve the problem of extra generation (e.g., both Multi-D and GLIGEN have the phenomenon of excessive generation of "carrots"), mainly due to the inhibition loss used in MIGC. The fourth row shows that MIGC can achieve accurate quantity control while other methods generate wrong quantities. The results in the fifth row show that when it comes to quantity control of multiple categories, stronger attribute control (e.g., it can avoid cat attributes from leaking to the dogs' region) makes MIGC achieve more accurate quantity control.

F. Details of Shading Aggregation Controller

Overview. Illustrated in Fig. 6, after obtaining the shading results $\mathbb{R}_s = \{\mathbf{R}_s^1, \ldots, \mathbf{R}_s^N, \mathbf{R}^{bg}, \mathbf{R}_{LA}\} \in \mathbb{R}^{(N+2,C,H,W)}$ and guidance masks $\mathbb{M} = \{\mathbf{M}^1, \ldots, \mathbf{M}^N, \mathbf{M}^{bg}, \mathbf{M}_{LA}\} \in$

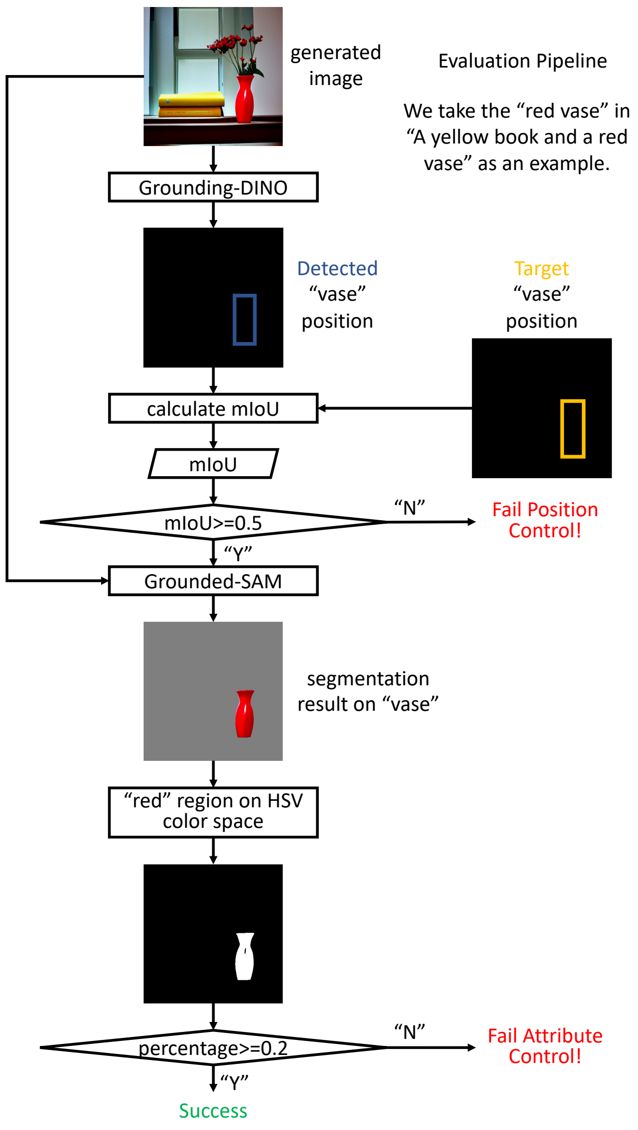

Figure 7. Evaluation Pipeline.

$\mathbb{R}^{(N+2,1,H,W)}$, the Shading Aggregation Controller (SAC) sequentially performs instance **intra-attention** and **inter-attention** to dynamically aggregate shading results, and aggregation weights summing to 1 are assigned to shading results on each spacial pixel through the softmax function, resulting in the final shading $\mathbf{R}_{final} \in \mathbb{R}^{(H,W,C)}$.

Instance Intra-Attention. As shown in Fig. 6, after SAC concats the shading results and guidance masks in the channel dimension, it will perform instance intra-attention through a stack of Conv-CBAM-Conv layers, in which Conv layers are mainly used to change the channel number, and the CBAM [19] sequentially performs channel-wise and spatial-wise attention in each instance's feature

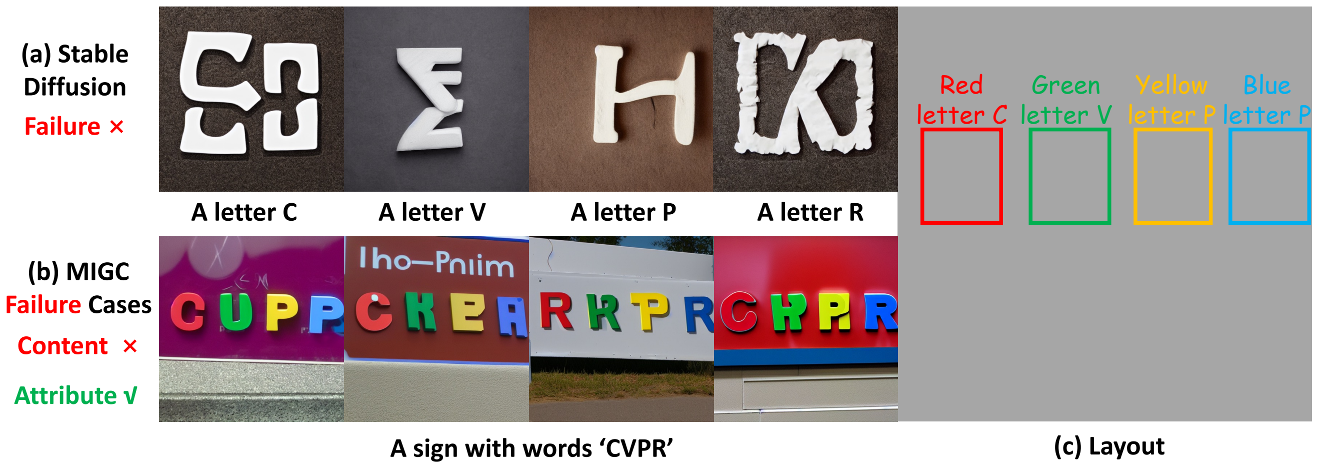

Figure 8. Limitation. (a) shows that stable diffusion struggles to generate individual letters 'C,' 'V,' 'P,' and 'R.' (b) shows the failure cases of MIGC. (c) shows the layout information of the failure case. If stable diffusion struggles to generate a specific instance, our MIGC will also encounter difficulties when generating this instance or its combination with other instances.

map.

Instance Inter-Attention. As shown in Fig. 6, after instance intra-attention, the SAC further performs instance inter-attention, in which the SAC reshapes the features to change the dimension order of the feature map and then uses the CBAM to perform instance-wise attention.

G. Details of Evaluation Pipeline

Overview. The flowchart in Fig. 7 shows the details of the evaluation pipeline, and we will introduce it by telling how to check whether "a red vase" is accurately generated.

Position Evaluation. First, we input the generated image into Grounding-DINO [14] to detect the bounding box of the "vase" and then calculate the IoU with the target layout's bounding box. If the IoU$\geq$0.5, this "vase" is determined to be *Position Correctly Generated*. Note that if multiple bounding boxes are detected in the generated image, we will select the one closest to the target layout's bounding box to calculate IoU.

Attribute Evaluation. After checking that the "a red vase" is *Position Correctly Generated*, we will further check whether its attribute (i.e., "red" color) is generated accurately. Specifically, we will use Grounded-SAM [11] to segment the "vase" region in the generated image and mark its area as M. Then we will calculate the area in M that meets the "red" requirement on the HSV color space and mark it as O. If the percentage O/M $\geq$ 0.2, we can consider that this "red vase" has the correct attribute and mark it as *Fully Correctly Generated*.

Evaluation for Different Benchmarks. Benchmarks requiring both attribute and position control, such as our proposed COCO-MIG, require each instance to be *Fully Correctly Generated*. Benchmarks requiring only position control, such as COCO-positon, require each instance to be *Position Correctly Generated*.

H. Manual Evaluation on DrawBench

We also perform a manual evaluation on DrawBench [18] to check whether the generated images adhere to the input text description in color, position, and count dimensions. Specifically, ten people will participate in the evaluation, and each generated image will be judged as "correctly generated" or "wrongly generated." We show the average accuracy calculated based on the evaluation results of ten people.

Different from automated evaluation, which strictly considers the mIoU to determine whether the local generation is successful or not, manual evaluation mainly checks whether the generated image satisfies the text description globally.

I. More Implementation Details

Training. We only deploy MIGC on the mid-layers (i.e., 8×8) and the lowest-resolution decoder layers of UNet(i.e., 16×16), which greatly determine the generated image's layout and semantic information [3, 15]. In other Cross-Attention layers, we use the global prompt for global shading. We use COCO 2014 [13] to train MIGC. To get the instance descriptions and their bounding boxes, we use stanza [17] to split the global prompt and detect the instances with the Grounding-DINO[14] model. To put the data in the same batch, we fix the number of instances to 6 during training, i.e., if data contains more than 6 instances, 6 of them will be randomly selected. If data contains less than 6 instances, we complete it with null text and coordinates [0.0, 0.0, 0.0, 0.0]. We train our MIGC based on the pre-trained stable diffusion v1.4. We use AdamW [10] optimizer with a constant learning rate of $1e^{-4}$, and train the model for 300

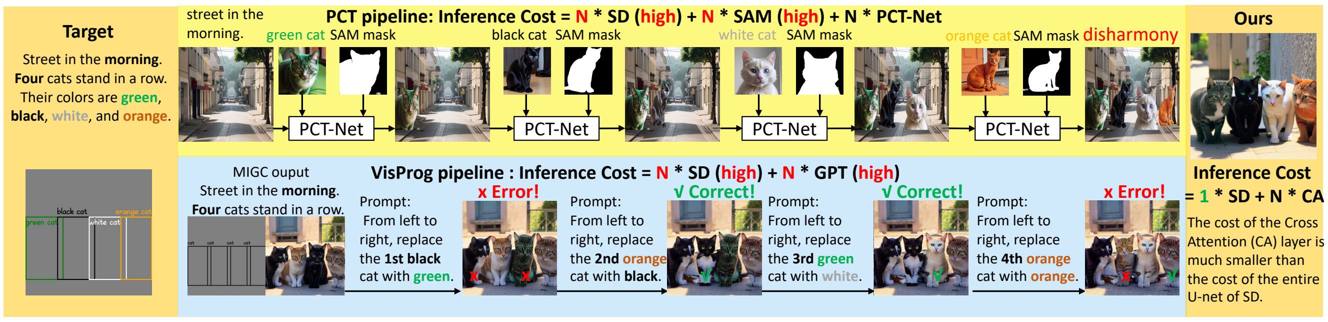
Target
Street in the morning. Four cats stand in a row. Their colors are green, black, white, and orange.
PCT pipeline: Inference Cost = N * SD (high) + N * SAM (high) + N * PCT-Net
street in the morning.
green cat
SAM mask
black cat
white cat
orange cat
disharmony
PCT-Net
Ours
MIGC ouput
Street in the morning. Four cats stand in a row.
VisProg pipeline : Inference Cost = N * SD (high) + N * GPT (high)
Prompt: From left to right, replace the 1st black cat with green.
x Error!
Prompt: From left to right, replace the 2nd orange cat with black.
√ Correct!
Prompt: From left to right, replace the 3rd green cat with white.
√ Correct!
Prompt: From left to right, replace the 4th orange cat with orange.
x Error!
Inference Cost = 1 * SD + N * CA
The cost of the Cross Attention (CA) layer is much smaller than the cost of the entire U-net of SD.

Figure 9. More baseline results. Please zoom in for the best view.

epochs with batch size 320, which requires 15 hours on 40 V100 GPUs with 16GB VRAM each.

Inference. We use EulerDiscreteScheduler [9] with 50 sample steps and use our MIGC in the first 25 steps. We select the CFG scale[8] as 7.5. As shown in Fig. 6, the channel number of the second CBAM layer (i.e., CBAM in Instance Inter Attention) in the Shading Aggregation Controller is related to the number of input instances. In order to allow our MIGC to handle different numbers of instances, we set the channel number of the second CBAM layer as max_num+2 (e.g., our default setting is 28+2, which can satisfy almost all practical applications). In actual inference, we assume that the number of instances to be processed is $n_{infer} \leq max\_num$, and we get $f_{intra} \in \mathbb{R}^{(1,n_{infer}+2,h,w)}$ through Instance Intra Attention. Next, we need to pad the number of channels of f_{intra} to max_num+2. Specifically, we enter an all-0 features $f_{zero} \in \mathbb{R}^{1,c,h,w}$ (i.e., this is consistent with the shading result of null text during the training) into Instance Intra Attention to get $f_{padding} = InstanceIntraAttenion(f_{zero}), f_{padding} \in \mathbb{R}^{1,1,h,w}$, and we use $f = concat([f_{intra}] + [f_{padding}] * (max\_num - n_{infer}), dim = 1), f \in \mathbb{R}^{1,max\_num+2,h,w}$ as the input of the CBAM layer in Instance Inter Attention. In order to allow the network to notice the later-ordered shading instances during actual inference, we will randomly shuffle the above $f_{padding}$ and f_{intra} during training, while the shading background and shading template will not participate in the above shuffle process. At the same time, we can observe that since Instance Intra Attention has eliminated the larger number of original feature channels C (e.g., 1280), the computational complexity will be very low even when processing a larger number of shading instances in Instance Inter Attention.

J. More Baselines

Based on the divide-and-conquer idea, we also designed two other baselines. The qualitative comparisons are shown in Fig.9

1)PCT-Net Pipeline. As shown in the first row of Fig.9, we first independently generate each individual instance and the background. Then, we use PCT-Net [6], a state-of-the-art image fusion network, to merge all instances with the background. Using PCT-Net to fuse the pre-generated images ensures the correctness of attributes, which verifies the effectiveness of our divide-and-conquer idea. However, this pipeline incurs significant inference time costs, and the generated images may lack harmony.

2)Visual Programming Pipeline. As shown in the second row of Fig.9, Visual Programming [7] utilizes the GPT model to parse user input commands and generate a series of predefined operations, thereby achieving functionalities such as image editing. Here, we employ this method to sequentially perform editing on each object in the pre-generated images from left to right, aiming to correct the attributes of each object as much as possible. This pipeline is capable of correcting erroneously generated attributes. However, it uses text to locate and edit the instance, which lacks precise positioning capabilities and faces challenges in deployment to real scenes with complex layouts. For example, the 1st and 4th steps locate and affect the incorrect cat. In addition, these methods utilize GPT to coordinate image generation, making large-scale generation expensive and challenging.

K. Limitation

Inspired by the idea of divide and conquer, MIGC maximizes the use of the powerful Single-Instance Generation capability of pre-trained stable diffusion and extends it to MIG tasks. However, for a specific instance that stable diffusion cannot generate well, our MIGC will also encounter difficulties when generating this instance or its combination with other instances. As Fig.8(a) shows, stable diffusion has difficulty generating individual letters accurately. Therefore, when using MIGC to generate the words ‘CVPR’ in the layout of Fig.8(c), we see that although MIGC correctly controls the color attribute of each letter, the content of the actual letters is wrong, causing the entire sample to fail, as shown in Fig.8(b).

References

[1] Omer Bar-Tal, Lior Yariv, Yaron Lipman, and Tali Dekel. Multidiffusion: Fusing diffusion paths for controlled image generation. *arXiv preprint arXiv:2302.08113*, 2023. 1

[2] Hila Chefer, Yuval Alaluf, Yael Vinker, Lior Wolf, and Daniel Cohen-Or. Attend-and-excite: Attention-based semantic guidance for text-to-image diffusion models, 2023. 1

[3] Minghao Chen, Iro Laina, and Andrea Vedaldi. Training-free layout control with cross-attention guidance. *arXiv preprint arXiv:2304.03373*, 2023. 1, 7

[4] Weixi Feng, Xuehai He, Tsu-Jui Fu, Varun Jampani, Arjun Reddy Akula, Pradyumna Narayana, Sugato Basu, Xin Eric Wang, and William Yang Wang. Training-free structured diffusion guidance for compositional text-to-image synthesis. In *The Eleventh International Conference on Learning Representations*, 2023. 1

[5] Weixi Feng, Wanrong Zhu, Tsu-jui Fu, Varun Jampani, Arjun Akula, Xuehai He, Sugato Basu, Xin Eric Wang, and William Yang Wang. Layoutgpt: Compositional visual planning and generation with large language models. *arXiv preprint arXiv:2305.15393*, 2023. 1

[6] Julian Jorge Andrade Guerreiro, Mitsuru Nakazawa, and Björn Stenger. Pct-net: Full resolution image harmonization using pixel-wise color transformations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5917–5926, 2023. 8

[7] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962, 2023. 8

[8] Jonathan Ho. Classifier-free diffusion guidance. *ArXiv*, abs/2207.12598, 2022. 8

[9] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models, 2022. 8

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. 7

[11] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. *arXiv:2304.02643*, 2023. 7

[12] Yuheng Li, Haotian Liu, Qingyang Wu, Fangzhou Mu, Jianwei Yang, Jianfeng Gao, Chunyuan Li, and Yong Jae Lee. Gligen: Open-set grounded text-to-image generation. *CVPR*, 2023. 1

[13] 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, 2015. 1, 7

[14] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023. 7

[15] Chong Mou, Xintao Wang, Jiechong Song, Ying Shan, and Jian Zhang. Dragondiffusion: Enabling drag-style manipulation on diffusion models. *arXiv preprint arXiv:2307.02421*, 2023. 7

[16] OpenAI. Gpt-4 technical report, 2023. 1

[17] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2020. 7

[18] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L. Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, Seyedeh Sara Mahdavi, Raphael Gontijo Lopes, Tim Salimans, Jonathan Ho, David Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. 2022. 1, 7

[19] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module, 2018. 6

[20] Jinheng Xie, Yuexiang Li, Yawen Huang, Haozhe Liu, Wentian Zhang, Yefeng Zheng, and Mike Zheng Shou. Boxdiff: Text-to-image synthesis with training-free box-constrained diffusion. *arXiv preprint arXiv:2307.10816*, 2023. 1