A Supplementary material

A.1 Detailed architecture

In this section, we provide additional implementation details about our architecture in order to consolidate the already-presented Fig. 3 (overview of the generators) and Fig. 4 (conditioning blocks).

One-step generator. In sec. 3.2 we mention that we use [31] as the backbone for the one-step model, and that we insert conditioning information in the normalization blocks as well as in the very first layer of the generator. In Fig. 8 (top) we show the detailed architecture of this model. The implementation of an individual "SPADE ResBlock" is specified in [31], but for reference we mention that each residual block consists of two normalization blocks wrapped by a skip-connection. If the number of input and output channels does not match, the skip-connection is learned, i.e. a third normalization block is learned. In the models conditioned on captions, we never attach attention inputs to skip-connections (to avoid potential instabilities). Each normalization block learns its own set of weights, and in our case they correspond to the S or S_{avg} blocks specified in Fig. 4.

Two-step generator. The architecture of the two-step generator is depicted in Fig. 9, and differs significantly from the aforementioned implementation. The background generator G_1 is a simplified version of the one-step generator with fewer residual blocks. The foreground generator G_2 implements a bottleneck architecture that takes as input the generated background image and compresses it through a series of *unconditional* residual blocks. The low-resolution feature-map is then expanded again through a series of *conditional* blocks. Interestingly, for foreground manipulations it is possible to preprocess the feature maps up to the last *unconditional* downsampling block in G_2 (8×8 resolution) and greatly speed up regeneration.

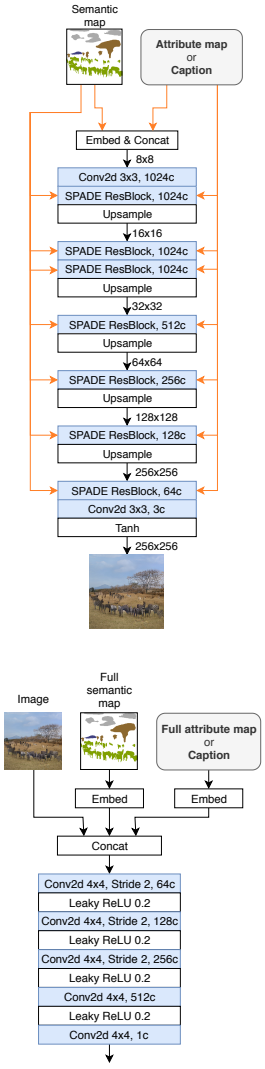

Fig. 8: **Top:** one-step generator using the SPADE backbone. "1024c" stands for "1024 output channels". The number on the right of an arrow specifies the feature map resolution at that level. Orange arrows indicate that the input information is fed to S blocks. **Bottom:** discriminator (used in all architectures).

Discriminator. We use the multi-scale discriminator from [31, 44] and change its input layer to add information about attributes or captions. The architecture is shown in Fig. 8 (bottom). As usual with multi-scale discriminators, we train two instances: one which takes as input an image at full resolution, and one which takes as input a downsampled version (by a factor of two). They learn different sets of embeddings and different sets of attention heads if the style is conditioned on a sentence.

Model complexity. Table 2 presents the number of parameters for all variants of our approach. The SPADE baseline trained on the 182 COCO-Stuff classes requires 97.5M parameters. Our 1-step baseline trained without style information (neither attributes nor captions) on our set of 280 classes requires a slightly lower number of parameters (94.2M) thanks to the pixel-wise class embeddings, even though the number of classes is larger. In the version with attributes, the added cost (+2.3M parameters) is only due to the learned attribute embeddings (256 64d embeddings per normalization block). In the version with captions, the custom attention modules add 12.5M parameters (for 6 heads) or 23.3M parameters (for 12 heads). The number of parameters can be easily tuned by varying the number of attention heads. We conduct a similar analysis on the two-step model. In this case, the background generator is slightly more powerful than the foreground generator.

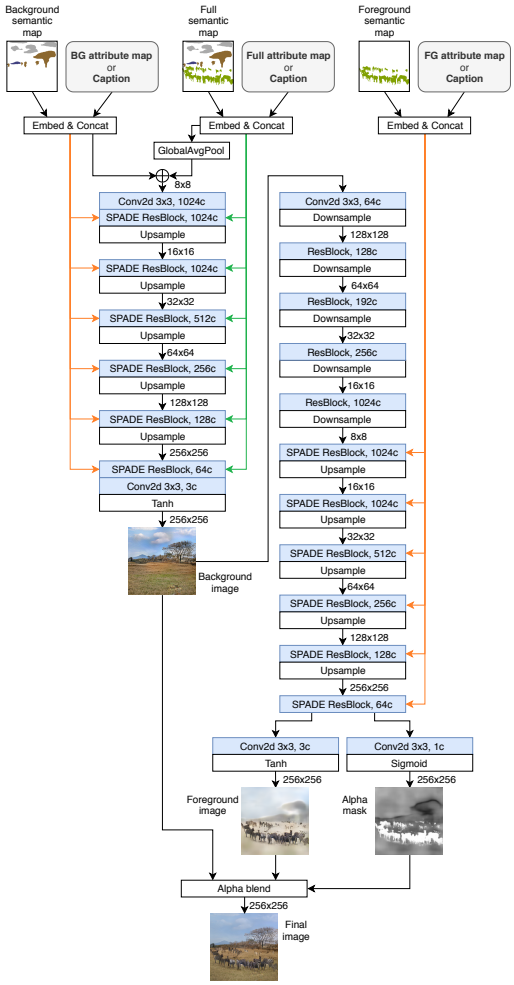

Fig. 9: Two-step generator. The left side of the figure depicts G_1 (background generator), while the right side depicts G_2 (foreground generator). Orange arrows indicate that the input information is fed to S blocks, whereas green arrows denote inputs to S_{avg} blocks.

Table 2: Number of parameters for different variations of our approach. For the two-step models we specify the numbers for both generators (respectively G_1 and G_2). "6h" denotes "6 attention heads".

Approach	Style input	# params
Baseline [31]	None	97.5M
1-step	None	94.2M
1-step	Attributes	96.5M
1-step	Text (6h)	106.7M
1-step	Text (12h)	117.5M
2-step	None	74.5M + 50.6M
2-step	Attributes	78.3M + 51.9M
2-step	Text (12h)	90.7M + 65.8M

Sparse map generation and manipulation. In this paragraph we provide further details in addition to those presented in sec. 4.1. Specifically, we describe how we construct and maintain the data structure that enables instance manipulation and rasterization into a sparse semantic map. Since a scene may consist of objects that partially overlap, the order in which they are drawn on the semantic map matters, e.g. given a *car* and its *headlight*, we want to render the headlight semantic mask on top of the car and not the opposite. Therefore, we sort all instances by mask area and draw them from the largest to the smallest. Additionally, we construct a scene graph to facilitate manipulation: if 70% of the area of an instance is contained within another instance, it becomes a child of the latter. With regard to the previous example, moving the car would also move the headlights attached to it. Finally, in our experiments on Visual Genome, we link attributes to an instance if the IoU between the ground-truth region and the detected bounding box is greater than 0.5.

Training details and hyperparameters. In all experiments, we train on 8 Pascal GPUs for 100 epochs using Adam (learning rate: 1e-4 for G, 4e-4 for D, one G update per D update), and start decaying the learning rate to 0 after the 50th epoch in a linear fashion. We use a batch size of 32 for the *one-step* model and 24 for the *two-step* model (the largest we can fit into memory), with synchronized batch normalization. Training takes one week for the one-step model and two weeks for the two-step model. For the alpha blending loss term, we start from a factor of 10, and decay it exponentially with $\alpha = 0.9997$ per weight update, down to 0.01. For the experiments with captions, since COCO comprises five captions per image, we randomly select one caption at training time. In the evaluation phase, we concatenate the representations of all captions since our attention model can easily decide which ones to attend to.

A.2 Additional inference details

Randomizing style. In sec. 4.2 we mention that we can randomize the style of an image by sampling attributes from a per-class empirical distribution. More precisely, we estimate a discrete probability distribution of the attributes assigned to each class of the dataset. This includes the empty set (no attribute

for a given instance) as well as compound attributes (e.g. *blue and red* is different than *blue* or *red*). At inference, for each instance, we sample an element from the distribution of the class to which the instance belongs. The two-step decomposition also allows us to specify different strategies for the background and foreground. In the examples in Fig. 7, all background instances of a given class take the same attributes as input (e.g. all trees are leafless), which results in scenes with coherent styles. Conversely, foreground instances are still fully randomized (it would not be realistic to see cars all of the same color, for example). Within an individual instance, the style of its children is uniform, e.g. the same attributes are assigned to all wheels of a car, but of course wheel styles can be different across different cars.

Interpolating style. Our approach allows for smooth interpolation of attributes and text. While attention models usually preclude interpolation (whereas models based on fixed-length sentence embeddings such as [50] easily allow it), our *sentence-semantic* attention mechanism enables interpolation over the *contextualized class embeddings*, i.e. over the pooled attention values. For all cases (masks, attributes, text), we respectively interpolate between class embeddings, attribute embeddings, and contextualized class embeddings using spherical interpolation (*slerp*), which traverses regions with a higher probability mass [22]. Unlike [50], we found it unnecessary to enforce a prior on the embeddings via

Fig. 10: Interpolating style between two sentences (top two rows) and two attributes (bottom row). The smooth transitions across multiple factors of variation (e.g. color and time of the day) suggest that our latent space is structured and does not require regularization. For instance, in the middle row, the bus color traverses the region of *orange* while interpolating between red and yellow, even though it is not explicitly instructed to do so. Additionally, the headlights of the bus become increasingly brighter.

a KL divergence term in the loss. We show some examples of interpolation in Fig. 10 as well as in the supplementary video (sec. A.7).

Generating one object at a time. To ensure that foreground objects do not affect each other in the two-step model, it may be interesting to generate them one-by-one. In our experiments we generate all foreground objects at once by running a single instance of G_2, motivated by the much lower computational cost and the observation that foreground objects are usually well-separated. Nonetheless, our framework is flexible enough to support one-by-one generation of objects. In this regard, G_2 can be run independently for each object, and the output images and masks can be combined into a single, final image. Denoting the background image as $\mathbf{x}_{\text{bg}}$, the foreground images as $\mathbf{x}_{\text{fg}}^{[i]}$ ($i \in \{1 \dots N\}$), and the corresponding *unscaled* (i.e. before the activation function) transparency masks as $\boldsymbol{\alpha'}_{\text{fg}}^{[i]}$, we can generalize Equation 1 as follows:

$$\mathbf{w}_{\text{fg}}^{[i]} = \text{softmax}_i \left(\boldsymbol{\alpha'}_{\text{fg}}^{[i]} \right) \tag{3}$$

$$\mathbf{x}_{\text{fg}} = \sum_i \mathbf{x}_{\text{fg}}^{[i]} \odot \mathbf{w}_{\text{fg}}^{[i]} \tag{4}$$

$$\boldsymbol{\alpha}_{\text{fg}} = \sum_i \text{sigmoid} \left(\boldsymbol{\alpha'}_{\text{fg}}^{[i]} \right) \odot \mathbf{w}_{\text{fg}}^{[i]} \tag{5}$$

$$\mathbf{x}_{\text{final}} = \mathbf{x}_{\text{bg}} \cdot (1 - \boldsymbol{\alpha}_{\text{fg}}) + \mathbf{x}_{\text{fg}} \cdot \boldsymbol{\alpha}_{\text{fg}} \tag{6}$$

The second line combines foreground images into a single image through an object-wise weighted average. The same is repeated for the transparency channel (third line). Finally, the alpha blending is performed as in Equation 1. This formulation is differentiable and can be used for training the model, although the memory requirement may be excessive in high-resolution settings.

A.3 FID evaluation

The FID metric is very sensitive to aspects such as image resolution, number of images (where a low number results in underestimated FID scores), and the weights of the pretrained Inception network. To be consistent with [31], we try to follow their methodology as closely as possible. We resize the ground-truth images to the same resolution as the generated ones (256×256), and we keep the two sets aligned, i.e. one generated image per test image. We use the weights of the pretrained InceptionV3 network provided by PyTorch. To make the results in Table 1 comparable, we retrained the baseline from [31] and evaluated the results using our methodology.

A.4 Additional results

Semantic and style manipulation. Fig. 11 and Fig. 12 show examples of semantic manipulation and style manipulation (either using attributes or text).

Fig. 11: Examples of semantic and attribute manipulations (Visual Genome dataset). The images are generated by our two-step model. In the first row, the background is frozen to encourage locality.

Fig. 12: Further examples of style manipulation using text (COCO validation set). It is possible to control the style of individual instances (albeit in a less targeted fashion than attributes) as well as the global style of the image.

The last row of Fig. 12 suggests that our attention mechanism can correctly exploit the contextualized token embeddings produced by BERT. For instance, the caption "a black and white cat" affects only the cat, while "a black and white picture of a cat" affects the entire scene by generating a black-and-white image.

Two-step model. Fig. 17 shows additional demos generated by our two-step model on the Visual Genome validation set. In particular, we highlight the decomposition of the background and foreground, and the inputs taken by G_1 and G_2. Since G_2 outputs a soft transparency channel for the alpha blending, it can slightly violate the constraints imposed by the *foreground mask*. This allows it to draw reflections and shadows underneath foreground objects. Furthermore, as we mention in sec. 3.1, the motivation behind the two-step generator is that it facilitates local changes. In Fig. 15 we qualitatively compare one-step and two-step generation when manipulations are carried out on the input conditioning information (mask and style). We show that, in the two-step model, local manipulations do not result in global changes of the output. To further enhance locality, the background can be frozen when manipulating the foreground.

Table 3: Comparison to layout-based methods. The metric is the FID score [13]; lower is better. "GT BBox" stands for "ground-truth bounding-box", whereas our approach uses the sparse masks inferred from an object detector as usual.

Approach	Input	Training set	Test set	FID
Sg2im [18]	GT BBox layout	COCO-train	COCO-val	67.96
Layout2im [52]	GT BBox layout	COCO-train	COCO-val	38.14
LostGAN [40]	GT BBox layout	COCO-train	COCO-val	34.31
Ours (#3)	Sparse mask	COCO-train	COCO-val	18.57
Ours (#5)	Sparse mask	VG+ (aug.)	COCO-val	**17.98**

Comparison with layout-based methods. While in sec. 4.2 we compare our approach to [31] under uniform settings, it is also interesting to see how our sparse mask setting compares to approaches that generate images from bounding-box layouts (which are also sparse by nature) [15, 40, 52]. While these methods address a harder task (bounding boxes provide less information than segmentation masks), their applicability has only been demonstrated in low-resolution settings (typically 64×64), which makes them not directly comparable to our higher-resolution setting. To our knowledge, no bounding-box approach can currently generate high-resolution images that have the same visual quality and geometric coherence as mask-based approaches. Nonetheless, for completeness, in Table 3 we compare our sparse mask approach to these layout-based methods. We use the models trained on COCO or VG+ with no style input (rows #3 and #4 in Table 1, left), and downscale our images to 64×64 before computing the FID score.

Qualitative comparison of input masks. In Fig. 16, we show qualitative results for different input masks, both in fully supervised and weakly supervised settings. Additionally, in the figure we show qualitative results for the *sparsified* COCO model (ablation I in Table 1, right), where we keep only the "thing"

classes of COCO. While the outputs produced by the semantic segmentation maps are satisfactory, it is not clear how to manipulate them as they present banding artifacts and jagged edges.

A.5 Attention visualization

The behavior underlying our attention model can be easily visualized. Our formulation (*sentence-semantic* attention) is particularly suited for visualization tasks because it is tied to the semantic map, and not to feature maps in inner convolutional layers. Therefore, for each class in the semantic map (e.g. *person*, *tree*, empty space), we can observe how the sentence conditions that particular class. Considering that the attention modules have multiple entry points in the generator (one for each normalization block), it is easier to carry out this analysis in the discriminator, where there are only two entry points (in the input layer of each discriminator, since we adopt a multi-scale discriminator). We select the first discriminator for illustration purposes, and show the resulting attention maps in Fig. 13. The figure shows what parts of the sentence the discriminator is attending to in order to discriminate whether the caption is suitable for the input image.

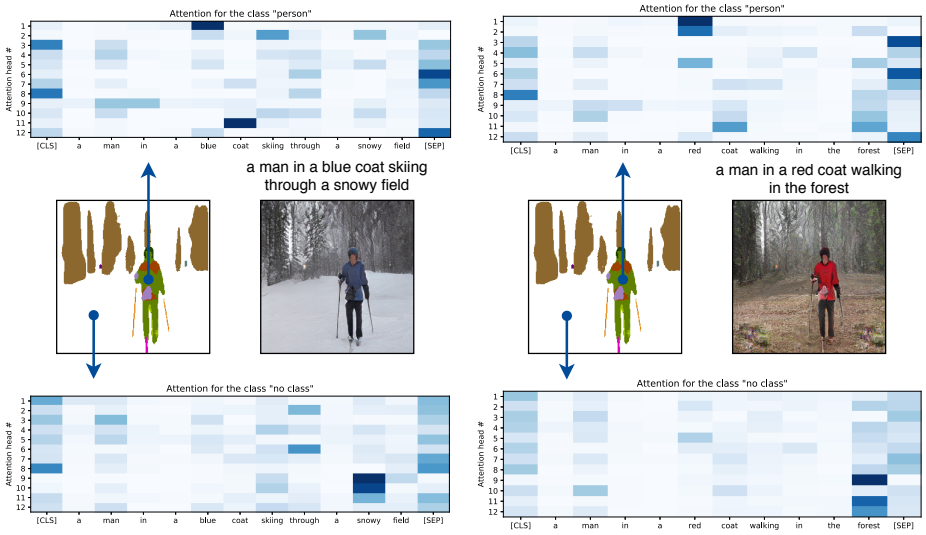

Fig. 13: Visualization of the attention mechanism in the discriminator for two images generated from the same semantic map, but different captions. An attention map is produced for each class in the semantic map, and each of these consists of 6 or 12 independent attention heads (12 here). In this illustration we only show those corresponding to *person* and *no class* (i.e. blank space) for clarity. `[CLS]` and `[SEP]` are special delimiters indicating respectively the start and end of a sentence. A head paying attention to these can be interpreted as not being triggered by the sentence. In the attention maps, a darker color indicates a higher weight.

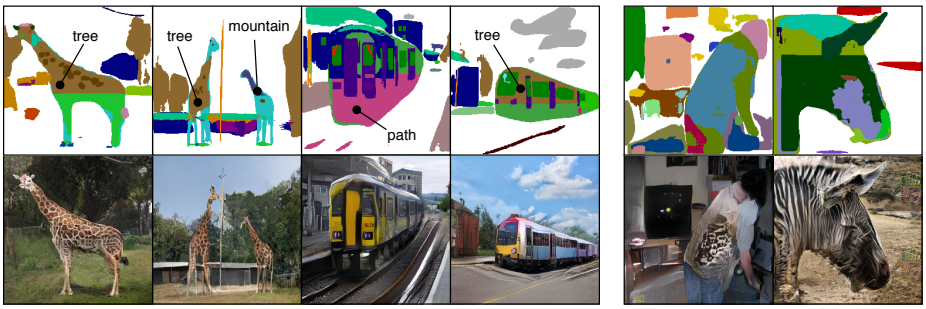

Fig. 14: **Left:** in many cases, weakly-supervised training leads to input noise robustness, i.e. artifacts in the input mask are not visible in the generated images. **Right:** some failure cases where the artifacts are visible in the output images.

A.6 Negative results

In this section, we discuss some of the unsuccessful ideas that we explored before reaching our current formulation.

Two-step model. Before successfully achieving two-step generation with sparse masks, we tried to implement the same idea using dense COCO segmentation maps. In the areas corresponding to foreground objects, G_1 (the background generator) would always render visible gaps. We tried to regularize the model using *partial convolutions* (a recently-proposed approach for infilling), but this did not have the desired effect. We also experimented with an attention mechanism where foreground areas were masked in G_1. While this was partly successful in filling the gaps, the model was very difficult to train and the final visual quality was considerably lower.

Discriminator architecture. We explored various ways of injecting conditional information in the discriminator. While SPADE uses input concatenation, recent GANs conditioned on classes [1, 49] use *projection discrimination* [29]. This idea led to marginally better FID scores, but we observed that the contour of generated objects would stick too close to the input mask, essentially resulting in a "polygonal" appearance. On the other hand, input concatenation allows the model to slightly deviate from the input mask, possibly resulting in a greater robustness to mask noise.

Hyperparameters. We tried to vary the design of SPADE blocks, e.g. by stacking more layers or using dilated convolutions. These ideas had a detrimental effect on the final result and we decided not to pursue them further.

A.7 Demo video

The accompanying video in the supplementary material illustrates examples of interactive manipulations. Among other things, we show how images can be generated from sketches as the user draws the masks, extra results from the two-step model (including comparisons with the one-step model with regard to local changes), and interpolations in the latent space of text and attributes.

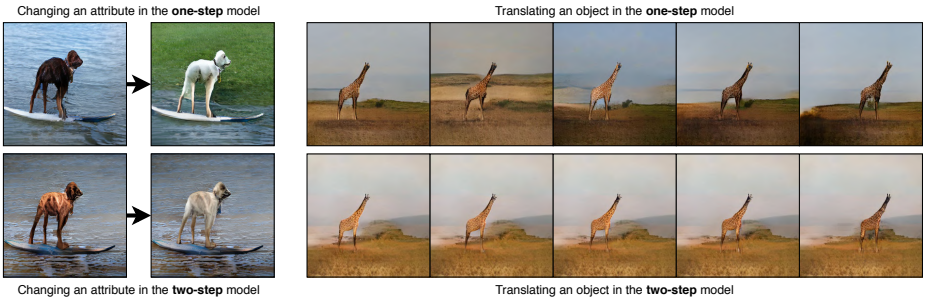

Fig. 15: In a single-generator model, local changes (e.g. changing the color of the dog to white) affect the scene globally due to learned correlations. The same can be observed when moving an object (e.g. left to right), as the representation space is discontinuous. In the two-step model, we can locally manipulate the background and foreground.

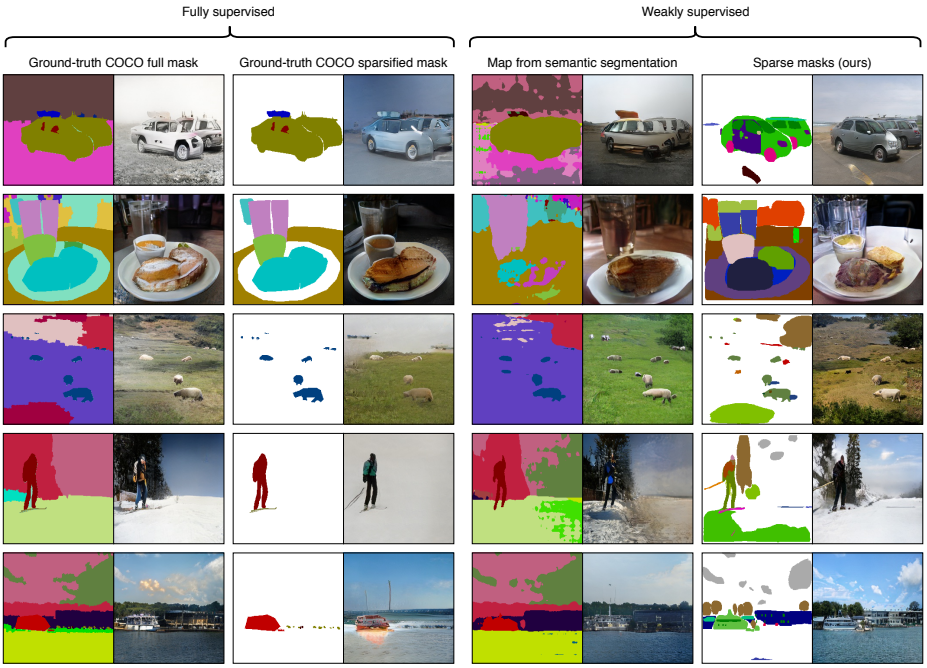

Fig. 16: Input masks for different approaches, and corresponding generated images. Our sparse masks do not present the typical artifacts of semantic segmentation outputs and are much easier to sketch or manipulate than dense maps.

Fig. 17: Demos generated by our two-step model. In addition to the full input mask, we show its decomposition into *background mask* and *foreground mask* (taken as input in S blocks respectively by G_1 and G_2). Note that G_1 also takes as input the full mask in S_{avg} blocks.