

# ObjBlur: A Curriculum Learning Approach With Progressive Object-Level Blurring for Improved Layout-to-Image Generation

## Supplementary Material

### 1 EXAMPLE OBJBLUR IMPLEMENTATION

Below we provide an example implementation of ObjBlur using PyTorch's `__getitem__()` method within the Dataset class. Most layout-to-image models process object annotations individually; we use this for-loop to generate  $\hat{\mathbf{x}}_{\text{LR} \rightarrow \text{HR}}$  and  $\hat{\mathbf{x}}_{\text{HR} \rightarrow \text{LR}}$ . Alternatively, binary masks defining the object regions can be pre-computed and used as described in main paper.

```
1 import torchvision.transforms as T
2
3 def __getitem__(self, index, t, T, start_res, p_obj):
4     """
5     Load an image from the dataset at given index, apply ObjBlur, and then return it.
6
7     Parameters:
8     index (int): The index of the image in the dataset.
9     t (float): The current training step.
10    T (float): The total number of training steps.
11    start_res (int): The start resolution to compute the low-resolution image.
12    p_obj (float): Probability threshold [0, 1] to apply blur to objects or background.
13
14    Returns:
15    tuple: A tuple containing the ObjBlur'ed image and the object data associated with the image.
16    """
17
18    # Load clean *square* image from the dataset
19    image_path = self.id_to_path(index)
20    hr_image = load_image(image_path)
21    hr_size = hr_image.shape[-1] # height == width
22
23    # Compute LR image resolution
24    blur_strength = (1 - t/T) # Linear CL schedule (1->0 for t:0->T)
25    lr_size = int((1 - blur_strength) * (hr_size - start_res) + start_res)
26
27    # Get blurred image by resizing to LR and then back to HR
28    down = T.Resize(lr_size)
29    up = T.Resize(hr_size)
30    lr_image = up(down(hr_image))
31
32    # Decide randomly whether to blur objects or background
33    blur_flag = random.randint(0, 100)
34
35    # Get object data for the current image
36    obj_data = self.id_to_objects(index)
37    for obj in enumerate(obj_data):
38        # Get bounding box coordinates for current object
39        x, y, w, h = obj['bbox']
40
41        # Apply object-level blurring based on blur_flag
42        if blur_flag <= p_obj * 100:
43            # Blurred objects: cut-and-paste LR objects into HR image
44            hr_image[:, y:y+h, x:x+w] = lr_image[:, y:y+h, x:x+w]
45        else:
46            # Blurred background: cut-and-paste HR objects into LR image
47            lr_image[:, y:y+h, x:x+w] = hr_image[:, y:y+h, x:x+w]
48
49    # Return the processed image and object data
50    if blur_flag <= p_obj * 100:
51        return hr_image, obj_data
52    else:
53        return lr_image, obj_data
```

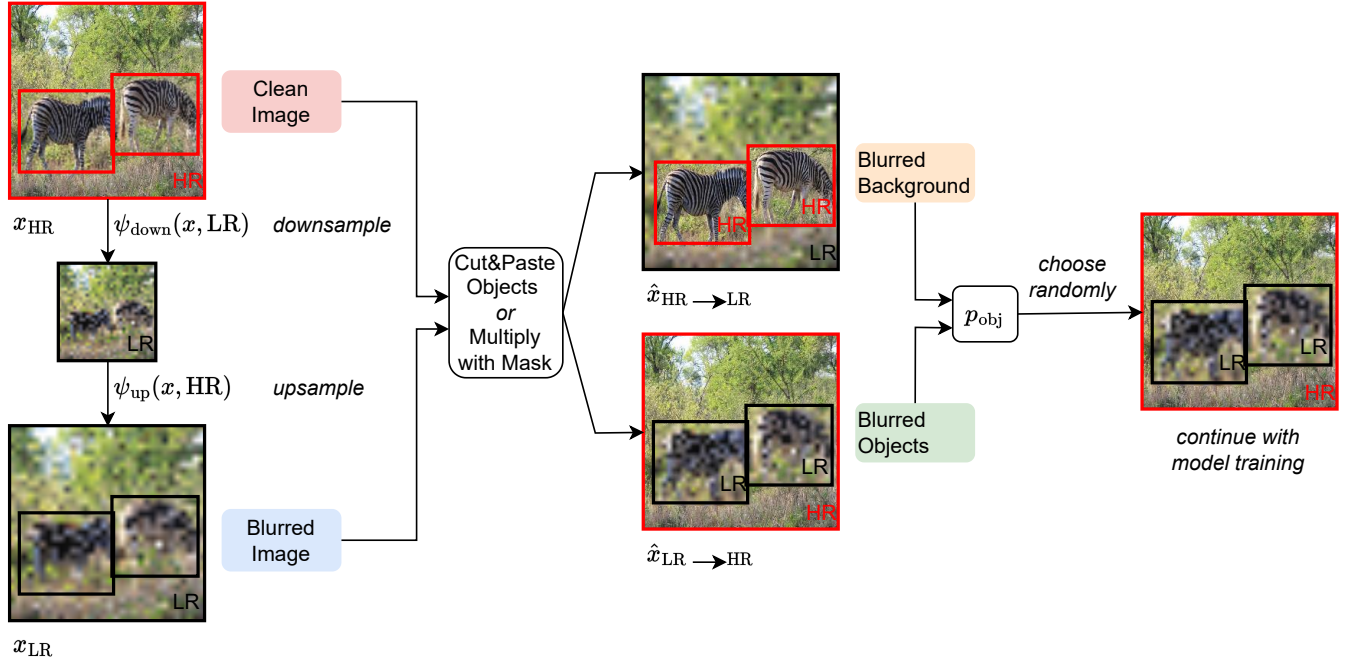


Figure 1: Visualization of our data loading process to create ObjBlur'd images as described in Listing 1. We start by sampling a clean image from the dataset and producing a blurry version via downsampling and subsequent upsampling. Next, we produce two new training samples by cut-and-pasting HR object regions into the LR image and vice versa. Finally, depending on  $p_{obj}$ , we randomly choose whether to return the sample with blurred objects or blurred background and continue with model training.

## 2 NEGATIVE RESULTS

We explored other novel and interesting techniques which ended up degrading or otherwise not improving performance or stability in our setting. We report them here for completeness, save time for future work and give a more complete overview of our attempts at improving layout-to-image models with curriculum learning approaches. However, these experiments and results are less thorough and specific to our particular setting. Thus, different perspectives, experimental settings, or implementations could still be fruitful research directions.

- **Class-Wise Curriculum Learning:** Not all object classes are equally difficult to learn. For example, there are many classes with a strong texture bias, such as “sky” and “grass” which are arguably much easier to learn than, for example, “person”. We sorted the classes using CAS (a proxy for generative difficulty) and experimented with an easy-to-difficult mechanism in the data loader. In other words, easier classes are learned first, and more complex classes are gradually added to the model. However, we only found slight improvements in SceneFID but a decrease in global image FID (even if combined with our blurring schedule).
- **Number of Objects Curriculum Learning:** With similar motivation, we also experimented with the number of objects within an image. Intuitively, generating 1 object per image should be easier than generating 10. This is further exacerbated through possibly complex relationships between individual objects and occlusion issues. To that end, we experimented with a CL schedule, where the number of object annotations per image gradually increases from 1 to N such that the network can learn incrementally to generate more objects. However, this approach did not succeed (even if combined with our blurring schedule). A potential reason is that many difficult object regions were initially treated as background.

## 3 MORE VISUAL RESULTS

We provide more visual results to compare the generated images with and without ObjBlur. Results on COCO are shown in Figure 2 and Figure 3. Results on Visual Genome are shown in Figure 4 and Figure 5. Images generated using our method are generally better. Objects are more recognizable and most often show similar or more details.



Figure 2: Visual comparison of generated images with and without using ObjBlur during training on COCO, Part 1/2.



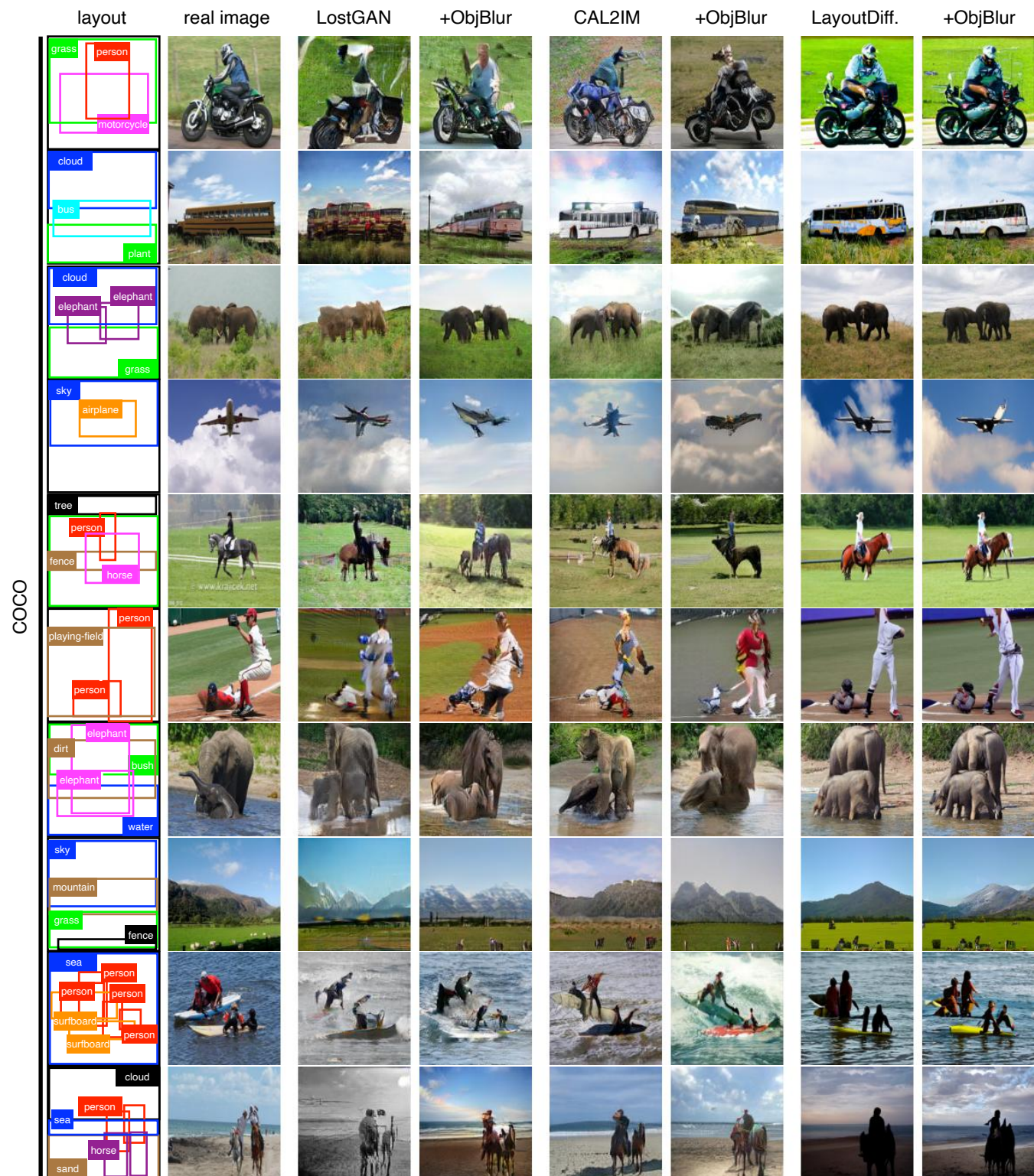


Figure 3: Visual comparison of generated images with and without using ObjBlur during training on COCO, Part 2/2.



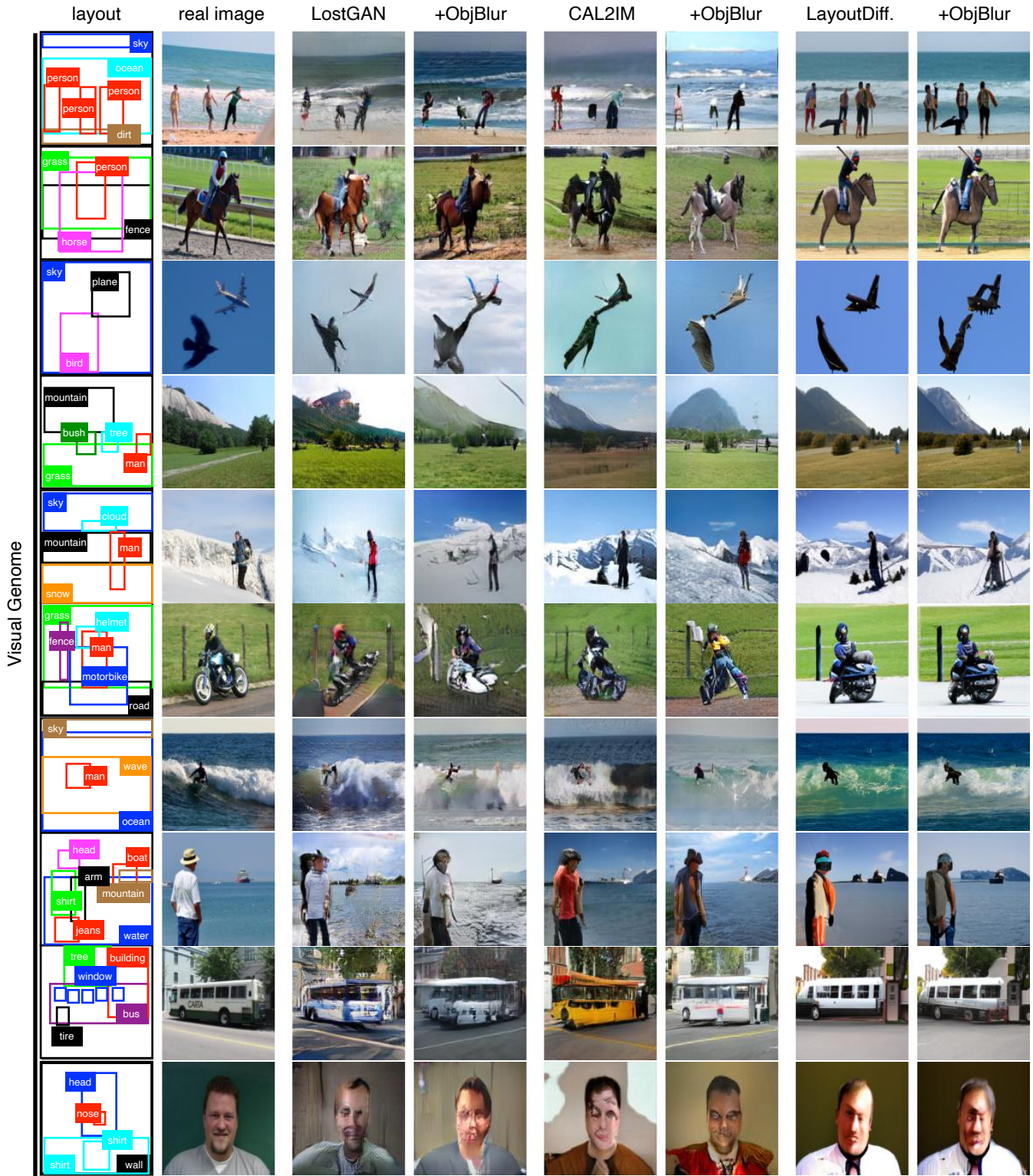


Figure 4: Visual comparison of generated images with and without using ObjBlur during training on VG, Part 1/2.



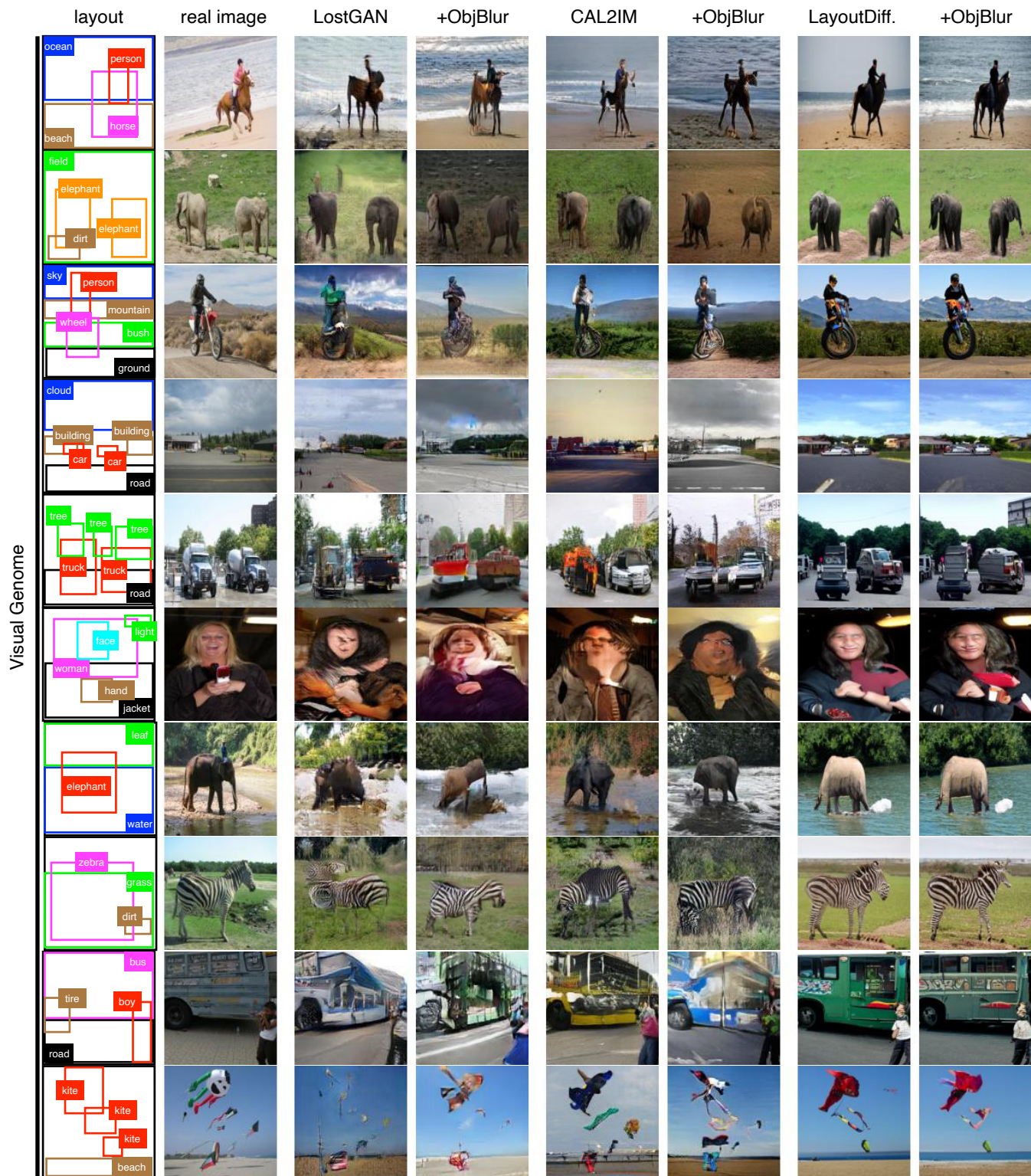


Figure 5: Visual comparison of generated images with and without using ObjBlur during training on VG, Part 2/2.