Adversarial Text to Continuous Image Generation
Supplemental Material

Kilichbek Haydarov Aashiq Muhamed Xiaoqian Shen Jovana Lazarevic

Ivan Skorokhodov Chamuditha Jayanga Galappaththige Mohamed Elhoseiny

{first_name.last_name}@kaust.edu.sa

King Abdullah University of Science and Technology

Contents

1. Experiment Details **1**

2. Inference, FLOPs and Memory Footprint **1**

3. Out-of-the-box Superresolution Generation **2**

4. More qualitative results **2**

1. Experiment Details

Dataset	#train	#validation	caption/image
MS-COCO	82k	40k	5
CUB	9k	3k	10
ArtEmis	60k	8k	5

Table 1. **Statistics of datasets.** The last column indicates the ratio of captions vs images.

Datasets. The statistics of datasets used to train the models are summarized in Table 1.

Hyper-parameters. In all our experiments, we set the hyper-parameters to predefined values: $\gamma = 5$, $\lambda = 10$ and $\tau = 0.5$ in loss functions. The rank hyper-parameter was fixed to value 5 for the INR backbone.

Implementation Details. Our models are trained with a learning rate $lr = 0.0025$ with multi-GPU support on 4 NVIDIA TESLA V100 GPUs. For all experiments, we kept the batch size equal to 16 and run for 25k iterations. Figure 1 shows the numpy-like pseudocode for the core implementation of our method.

2. Inference, FLOPs and Memory Footprint

We provide additional stats about our proposed model in tersm of Inference time (Inf. time), FLOPs, Memory footprint (Mem. Foot.) and compare them against other models. Table 2 shows that in terms of inference time our INR-based method is comparable to other methods. In our backbone, pixel generation is conducted independently, enabling a single forward pass through a Multi-Layer Perceptron (MLP) to evaluate RGB values for all pixels simultaneously. This parallel processing significantly mitigates concerns about latency in our HyperCGAN models. In the context of FLOPs, we are referring to

```python
def tensor_modulation_word(weight: Tensor, factors: Tensor):
    """
    Tensor modulation for word-level attention.

    Args:
        weight: [batch_size, c_out, c_in] - original weight
        factors: [batch_size, num_words, c_out + c_in]
    Returns:
        M: [batch_size, c_out, c_in] - modulating weight
    """

    b, c_out, c_in = weight.shape
    n_words = factors.shape[1]

    # Perform low-rank decomposition
    M = factors[:, :, :c_out].view(b, n_words, c_out, 1) \
        * factors[:, :, c_out:].view(b, n_words, 1, c_in) # [batch_size, num_words, c_out, c_in]

    # Compute attention
    M = M.view(b, n_words, -1)         # [batch_size, num_words, c_out * c_in]
    weight = weight.view(b, 1, -1)     # [batch_size, 1, c_out * c_in]
    score = torch.bmm(weight, M.transpose(1,2)) / np.sqrt(c_out * c_in) # [batch_size, 1, num_words]
    attn = F.softmax(score, -1) # [batch_size, 1, num_words]
    M = torch.bmm(attn, M) # [batch_size, 1, c_out * c_in]
    M = M.view(b, c_out, c_in) # [batch_size, c_out, c_in]

    M = M / M.norm(float('inf'), dim=[1, 2], keepdim=True)

    return M
```

Figure 1. Numpy-like pseudocode for word-level modulation mechanism

Method	Inf. time (sec)	FLOPs	Mem. Foot. (GB)
AttnGAN	0.016	261.73	1.18
ControlGAN	0.027	491.8	1.31
DM-GAN	0.017	261.9	1.18
DF-GAN	0.008	15.02	0.92
Lafite	0.02	90.28	1.29
VQ-DIff	9.83	569.5	3.13
HyperCGAN	0.019	11.93	1.24

Table 2. Additional stats in terms of Inference time, FLOPs and Memory Footprint. In our backbone, pixel generation is conducted independently, enabling a single forward pass through a Multi-Layer Perceptron (MLP) to evaluate RGB values for all pixels simultaneously. This parallel processing significantly mitigates concerns about latency in our HyperCGAN model.

”Floating Point Operations,” representing the total number of floating-point operations required for a single forward pass. A higher FLOPs count typically correlates with slower model performance and lower throughput. To quantify FLOPs, we employed the DeepSpeed FLOPs profiler for all the models discussed below. Furthermore, in assessing memory footprint, we considered the memory space occupied by the model's tensors and reported the estimated GPU memory consumption after model initialization. These measurements were consistently conducted on the V100 with 32GB memory.

3. Out-of-the-box Superresolution Generation

In this setting, we conduct out-of-the-box generation in a training efficiency manner naturally inherited from INR-based model. We first train our model on downsampled images (128×128) and perform 256×256 generation during inference either with classical interpolation methods or taking advantage of INR-based model's merits. This means our model can generate higher resolution images without modifying any architecture or finetuning, by just adjusting to a more denser coordinate grid. Table 3 show that our model outperforms standard upsampling techniques on all datasets. See Figure 2 for qualitative results.

4. More qualitative results

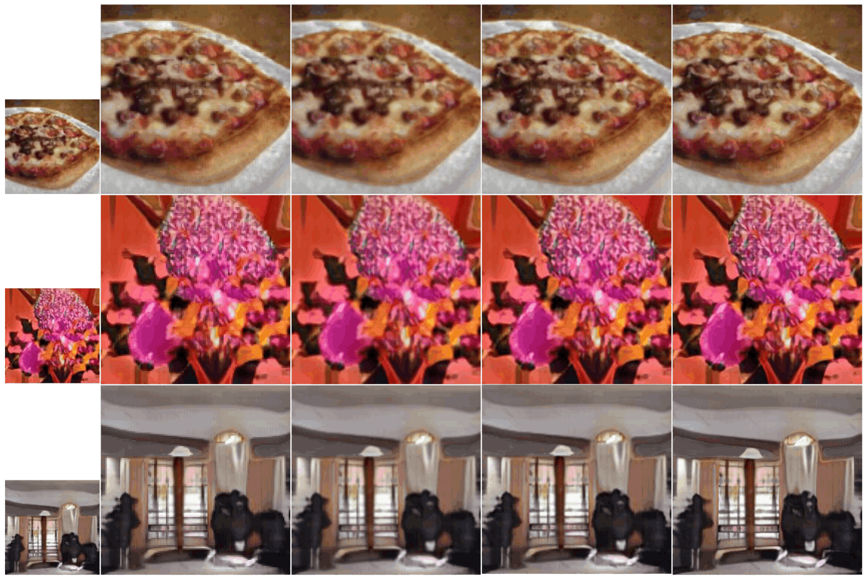

Figure 2. Qualitative comparison between classical interpolation techniques and our model.

Methods	COCO 256^2	ArtEmis 256^2	CUB 256^2
Nearest	30.86	26.87	17.69
Bilinear	29.84	28.06	16.84
Bicubic	28.73	26.52	16.18
HyperCGAN	**27.61**	**23.78**	**15.42**

Table 3. Super-resolution Synthesis comparison on FID scores. In this setting, we trained models on downsampled 128^2 images and generate 256^2 resolution images without changing architecture or finetuning.

Figure 3. More Qualitative Results of HyperCGAN on COCO and CUB datasets with resolution of 256^2.

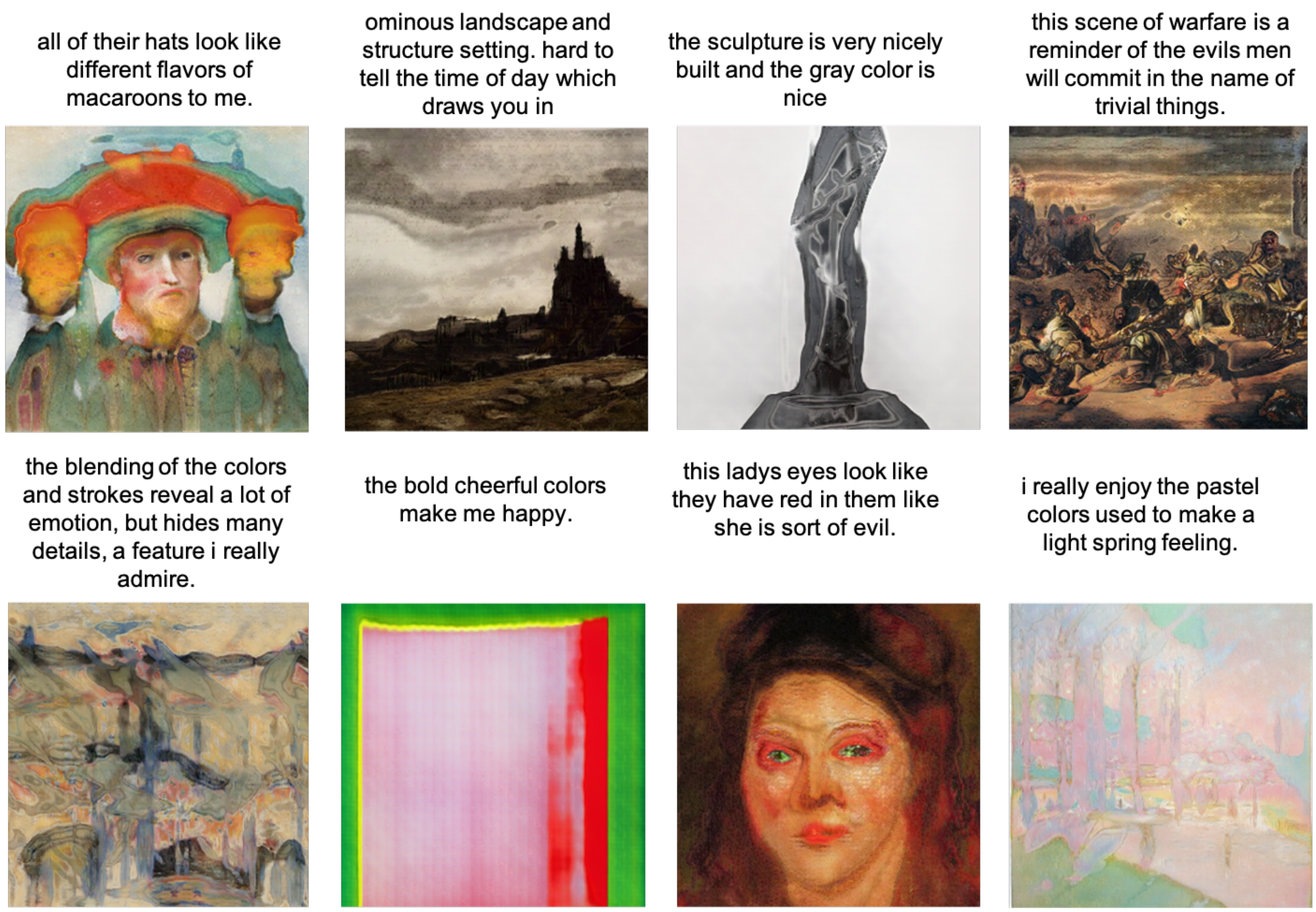

Figure 4. More Qualitative Results of HyperCGAN on ArtEmis.

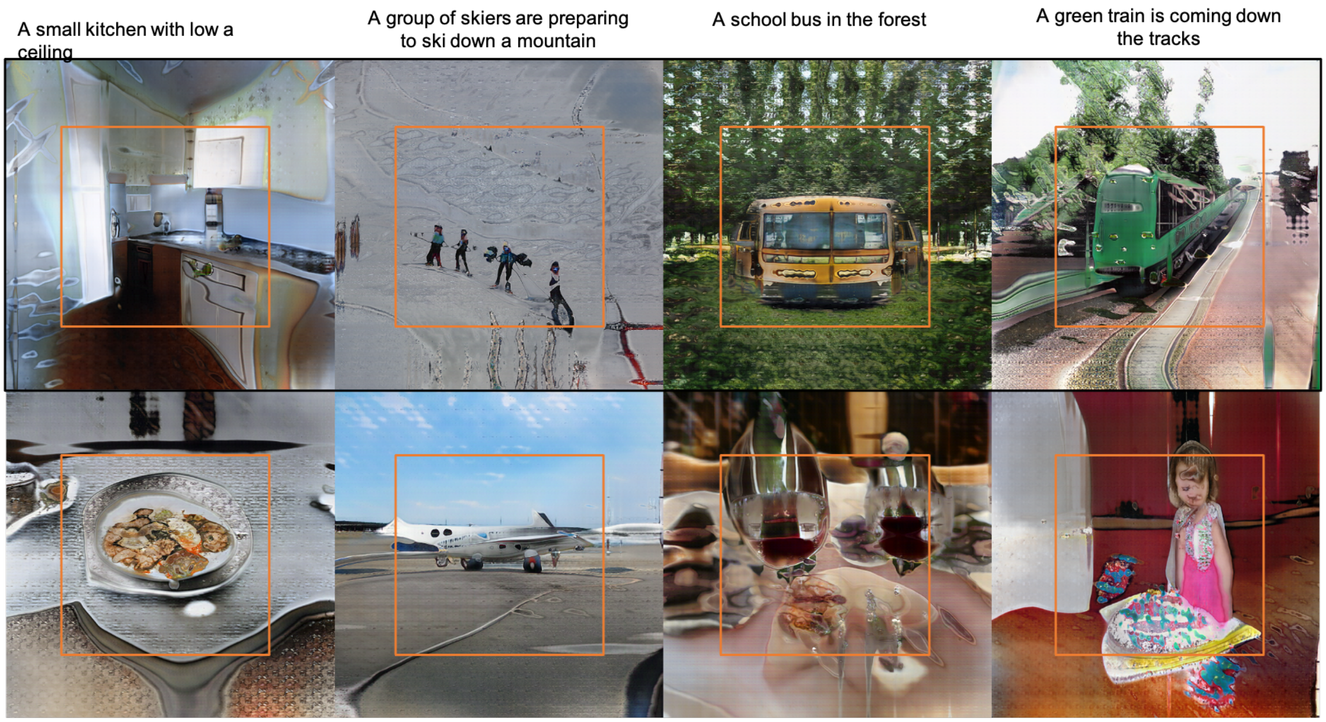

Figure 5. Extrapolation results of HyperCGAN on COCO with input coordinate range [-1.25, -1.25]

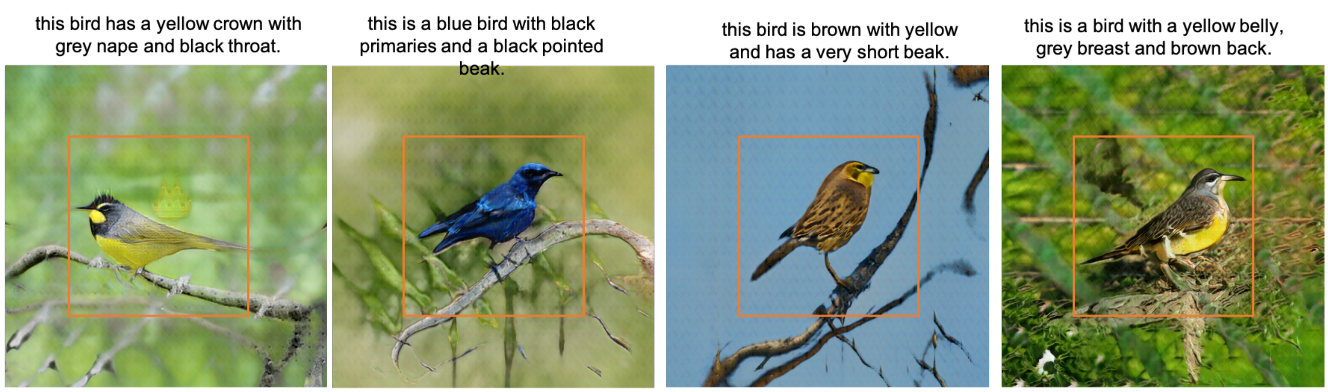

Figure 6. Extrapolation results of HyperCGAN on COCO with input coordinate range [-1.5, -1.5]

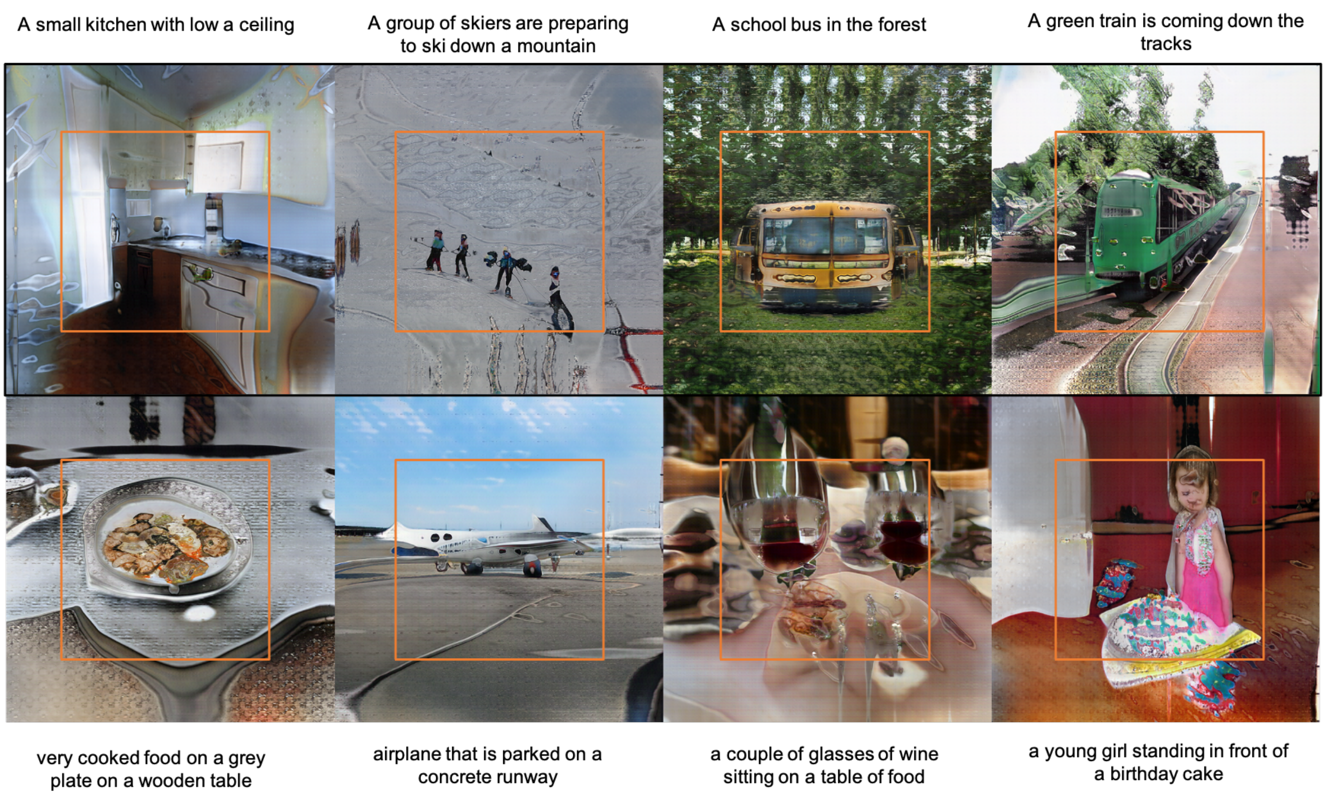

Figure 7. More Qualitative Results of HyperCGAN on Superresolution: model trained on 256^2 resolution and image synthesis is done on 1024^2.