## A. Factor graph and Bethe approximation

**Approximations** The Bethe approximation is a popular technique used in variational inference, probabilistic graphical models, and density estimation. One of the key pillars of Bethe approximation is approximating distribution entropy with Bethe entropy [29]. In terms of the factor graph,

$$\mathcal{H}_B := \sum_{j=1}^{m} \mathcal{H}_f^{(j)} + \sum_{i=1}^{n} (1 - d_i) \mathcal{H}_{\boldsymbol{x}}^{(i)}, \tag{8}$$

where $\mathcal{H}_f^{(j)}, \mathcal{H}_{\boldsymbol{x}}^{(i)}$ denote entropy of marginal distribution $q(f^{(j)}), q(\boldsymbol{x}^{(i)})$ defined over factor nodes and variable nodes respectively, that is,

$$\mathcal{H}_f^{(j)} = \int_{f^{(j)}} -q(f^{(j)}) \log q(f^{(j)}) df^{(j)}, \quad \mathcal{H}_{\boldsymbol{x}}^{(i)} = \int_{\boldsymbol{x}^{(i)}} -q(\boldsymbol{x}^{(i)}) \log q(\boldsymbol{x}^{(i)}) d\boldsymbol{x}^{(i)}.$$

In fact, Eq. (8) can be derived via the joint distribution approximation by its marginal distribution,

$$\int_{\boldsymbol{u}} -p(\boldsymbol{u}) \log p(\boldsymbol{u}) d\boldsymbol{u} = \sum_{j=1}^{m} \int_{f^{(j)}} -q(f^{(j)}) \log q(f^{(j)}) df^{(j)} + \sum_{i=1}^{n} (1 - d_i) \int_{\boldsymbol{x}^{(i)}} -q(\boldsymbol{x}^{(i)}) \log q(\boldsymbol{x}^{(i)}) d\boldsymbol{x}^{(i)}. \tag{9}$$

Approximations Eq. (8) are exact when the factor graph is an acyclic graph while practitioners also use approximations in general graphs where loops appear in the graph.

**Expressiveness of `DiffCollage`** The expressiveness of $p_\theta$ heavily depends on the underlying graphical model, which encodes the conditional independence structure of data in the sparse graph structure. A sparser graph results in a simpler joint distribution from which it is easier to draw samples. The sparsest graph consists of only variable nodes and no factor nodes, which indicates that all random variables are independent. Though it is the simplest joint distribution, such models fail to capture correlations between random variables and cannot express common distributions in the real world. On the other hand, while the fully connected graphs posses rich representation ability, they are difficult to infer or generate samples from, and Bethe approximation may suffer from large bias. Empirically, `DiffCollage` is expressive enough to approximate complex distributions on real datasets with different modalities.

**Hierarchical factor graph** Though we only present several simple factor graphs in the main paper, we can construct more expressive graphical models with hierarchical factor graphs. The idea behind a "hierarchical" factor graph is to treat the nodes and factors themselves as joint distributions over multiple random variables; by modeling these nodes / factors (defined over multiple random variables) with factor graphs, and their likelihood can still be evaluated with Bethe approximation.

The generation of 360-degree images is a good example, where each factor node in 360 cubemap graphis a smaller factor graph. Concretely, let us denote 6 faces as $\boldsymbol{x}^{(\mathrm{F})}, \boldsymbol{x}^{(\mathrm{B})}, \boldsymbol{x}^{(\mathrm{L})}, \boldsymbol{x}^{(\mathrm{R})}, \boldsymbol{x}^{(\mathrm{U})}, \boldsymbol{x}^{(\mathrm{D})}$. Then the three factor nodes follow $f^{(1)} = \{\boldsymbol{x}^{(\mathrm{F})}, \boldsymbol{x}^{(\mathrm{B})}, \boldsymbol{x}^{(\mathrm{L})}, \boldsymbol{x}^{(\mathrm{R})}\}$, $f^{(2)} = \{\boldsymbol{x}^{(\mathrm{F})}, \boldsymbol{x}^{(\mathrm{B})}, \boldsymbol{x}^{(\mathrm{U})}, \boldsymbol{x}^{(\mathrm{D})}\}$, $f^{(3)} = \{\boldsymbol{x}^{(\mathrm{L})}, \boldsymbol{x}^{(\mathrm{R})}, \boldsymbol{x}^{(\mathrm{U})}, \boldsymbol{x}^{(\mathrm{D})}\}$. The three variable nodes are $\boldsymbol{x}^{(1)} = \{\boldsymbol{x}^{(\mathrm{L})}, \boldsymbol{x}^{(\mathrm{R})}\}$, $\boldsymbol{x}^{(2)} = \{\boldsymbol{x}^{(\mathrm{U})}, \boldsymbol{x}^{(\mathrm{D})}\}$, $\boldsymbol{x}^{(3)} = \{\boldsymbol{x}^{(\mathrm{F})}, \boldsymbol{x}^{(\mathrm{B})}\}$. We can apply Bethe approximation for the factor graph over $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}$ and $f^{(1)}, f^{(2)}, f^{(3)}$. For the individual nodes and factors, the likelihood is defined over a set of random variables that can be modeled with factor graphs again, this time over the 6 faces. For the factor nodes, we treat it as a loop graph; for the variable nodes, as the faces contained within are opposite to each other, the corresponding factor graph would be just the two disconnected components.

Another way to improve expressiveness is to incorporate different conditional signals in different nodes of the factor graph. The approach can be interpreted as another type of hierarchical factor graph, which involves latent codes based on different conditional signals. In fact, several works [31, 53] that generate different patches of large images independently based on global code can be viewed as a hierarchical factor graph with latent code. We apply similar techniques in our conditional generation tasks, such as text-conditioned motion generation, and segmentation-conditioned image generation.

For applications that involve more complex dependencies among random variables and demand difficult inference tasks, more general graph representation, such as Junction tree [29], may have some advantages over representations based on factor graphs. We leave the generalization of `DiffCollage` to more complicated graphs for future research.

## B. Training and Sampling of DiffCollage

### B.1. Training

DiffCollage demands diffusion models over different pieces of the target content. Ideally, DiffCollage can work out of the box if diffusion models over each node in the factor graphs are available. When pre-trained models are not accessible, we can train DiffCollage in the same way as training standard diffusion models. We list the training algorithm in Algorithms 1 and 2. We note that the learning process of one marginal is independent of others, making the training procedure easy to scale since different marginals of DiffCollage can be learned in parallel. Moreover, different variable nodes or factor nodes may share the same diffusion models due to symmetry, improving the scalability further.

**Algorithm 1** Diffuison Collage: Training

**Inputs**: Marginal data on factor node $\{\mathcal{D}[f^{(j)}]\}$, marginal data on variable node $\{\mathcal{D}[i]\}$
**Output**: Score models $\boldsymbol{s}_\theta$ for marginal distributions
# Training for marginals can be conducted in parallel.
**for** $j \in 1, 2, \cdots, m$ **do**
  Training diffusion model $\boldsymbol{s}_\theta(f^{(j)}, t)$ on data $\mathcal{D}[f^{(j)}]$
**end for**
**for** $i \in 1, 2, \cdots, n$ **do**
  Training diffusion model $\boldsymbol{s}_\theta(\boldsymbol{x}^{(i)}, t)$ on data $\mathcal{D}[i]$
**end for**

**Algorithm 2** Training diffusion models for one node

**Inputs**: Marginal data $\mathcal{D}$
**Output**: Score models $\boldsymbol{s}_\theta$
**repeat**
  Sample $\boldsymbol{u}_0$ from $\mathcal{D}$
  Sample $t$ and Gaussian noise $\epsilon$
  $\boldsymbol{u}_t = \boldsymbol{u}_0 + \sigma_t \epsilon$
  Gradient descent on $\nabla_\theta[\omega(t) \left\| \nabla_{\boldsymbol{u}_t} \log q_{0t}(\boldsymbol{u}_t|\boldsymbol{u}_0) - \boldsymbol{s}_\theta(\boldsymbol{u}_t, t) \right\|^2]$
**util** converged

### B.2. Sampling

After training diffusion models for each marginal, DiffCollage implicitly obtains $p_\theta(\boldsymbol{u}, t)$ by its score $\nabla \log p_\theta(\boldsymbol{u}, t)$. The score of the learned distribution can be composited with its marginal scores $\boldsymbol{s}_\theta(\boldsymbol{x}^{(i)}, t), \boldsymbol{s}_\theta(f^{(j)}, t)$:

$$\nabla \log p_\theta(\boldsymbol{u}, t) = \boldsymbol{s}_\theta(\boldsymbol{u}, t) = \sum_{j=1}^{m} \boldsymbol{s}_\theta(f^{(j)}, t) + \sum_{i=1}^{n} (1 - d_i) \boldsymbol{s}_\theta(\boldsymbol{x}^{(i)}, t). \tag{10}$$

The marginal scores can be computed in parallel over the entire large content, which would significantly reduce the latency of the algorithm. We can plug Eq. (10) into existing diffusion model sampling algorithms. We include a deterministic sampling algorithm in Algorithm 3 for reference, though we re-emphasize that any sampler applicable to regular diffusion models would work with DiffCollage. Besides, DiffCollage also inherits the versatility of diffusion models and allows controllable generation without re-training, such as inpainting and super-resolution [8, 26]. We include more details regarding training-free conditional generation in Appendix C.1.

**Algorithm 3** `DiffCollage`: Sampling with Euler

**Inputs**: Score models $\boldsymbol{s}_\theta$, decreasing time steps $\{t_k\}_{k=0}^K$
**Output**: Samples from $p_\theta(\boldsymbol{u})$
Sample $\boldsymbol{u}_K$ from prior distribution $\mathcal{N}(0, \sigma_{t_K}\boldsymbol{I})$
**for** $k \in K, K-1, \cdots 1$ **do**
  # Pieces of $\boldsymbol{s}_\theta(\boldsymbol{u}_k, t_k)$ can be evaluated in parallel.
  $\boldsymbol{u}_{k-1} = \boldsymbol{u}_k + \dot{\sigma}_{t_k}\sigma_{t_k}\boldsymbol{s}_\theta(\boldsymbol{u}_k, t_k)(t_k - t_{k-1})$
**end for**
Return $\boldsymbol{u}_0$

# C. Experiments details

## C.1. Replacement and Reconstruction Methods for Conditioning

Here, we describe the details of **replacement** and **reconstruction** methods that are compared with `DiffCollage` in the experiments. In both cases, we are provided with an extra condition $\boldsymbol{y}$, and our goal is to generate $\boldsymbol{u}$ such that $\boldsymbol{y} = H(\boldsymbol{u})$ for some known function $H$. For example, $H : \mathbb{R}^n \to \mathbb{R}^m$ can be a low-pass filter that produces a low-resolution image (dimension $m$) from a high-resolution image (dimension $n$), and the task would essentially become super-resolution; similarly, one could define an inpainting task where $H$ is taking a subset of the pixels of the image $\boldsymbol{x}$. Diffusion models are particularly better-suited to such inverse problems than other generative models, such as GANs [41], as they can produce good results with much fewer iterations [26].

Both replacement and reconstruction methods make some modifications to the sampling procedure. At a high level, the replacement method makes a prediction over the clean image (denoted as $\hat{\boldsymbol{u}}_0$), and replaces parts of the image $\hat{\mathbf{u}}_0$ using information about $\boldsymbol{y}$; one could implement this as a projection if $H \in \mathbb{R}^{m\times n}$ is a matrix, *i.e.*, $\text{proj}(\hat{\boldsymbol{u}}_0) = H^\dagger\boldsymbol{y} + (I - H^\dagger H)\hat{\boldsymbol{u}}_0$ where $H^\dagger$ is the pseudoinverse of $H$. This is the strategy used in ILVR [8] and DDRM [26]. The reconstruction method, on the other hand, takes an additional gradient step on top of the existing sampling step that minimizes the $L_2$ distance between $\boldsymbol{y}$ and $H\hat{\mathbf{u}}_0$; this has been shown to produce higher-quality images than replacement methods on super-resolution and inpainting [9]. We describe the two types of conditional sampling algorithms in Algorithm 4 and Algorithm 5, respectively, using `DiffCollage`. This is almost identical to the conditional sampling algorithms with a standard diffusion model, as we only changed the diffusion to the one constructed by `DiffCollage`. For autoregressive baselines, we use these algorithms with regular diffusion models; for inpainting experiments with large images, we use them with `DiffCollage`.

**Algorithm 4** Replacement-based Conditioning using Regular Diffusion Models

**Inputs**: Observation $\boldsymbol{y}$, matrix $H$, score models $\boldsymbol{s}_\theta$, decreasing time steps $\{t_k\}_{k=0}^K$, sampling algorithm from time $t$ to time $s$ using a score function, denoted as $\text{sample}(\text{score}, \boldsymbol{u}_t, t, s)$.
**Output**: Samples from $p_\theta(\boldsymbol{u})$ where $\boldsymbol{y} = H(\boldsymbol{u})$
Sample $\boldsymbol{u}_K$ from prior distribution $\mathcal{N}(0, \sigma_{t_K}\boldsymbol{I})$
**for** $k \in K, K-1, \cdots 1$ **do**
  # Obtain denoising result from score function $\boldsymbol{s}_\theta(\boldsymbol{u}_k, t_k)$.
  $\hat{\boldsymbol{u}}_0 = \boldsymbol{u}_k + \sigma_{t_k}^2\boldsymbol{s}_\theta(\boldsymbol{u}_k, t_k)$.
  # Replacement projection based in $\boldsymbol{y}$ and $H$.
  $\tilde{\boldsymbol{u}}_0 = H^\dagger\boldsymbol{y} + (I - H^\dagger H)\hat{\boldsymbol{u}}_0$.
  # Sample based on corrected result.
  $\tilde{\boldsymbol{s}} = (\tilde{\boldsymbol{u}}_0 - \boldsymbol{u}_k)/\sigma_{t_k}^2$.
  $\boldsymbol{u}_{k-1} = \text{sample}(\tilde{\boldsymbol{s}}, \boldsymbol{u}_{t_k}, t_k, t_{k-1})$.
**end for**
Return $\boldsymbol{u}_0$

**Algorithm 5** Reconstruction-based Conditioning with `DiffCollage` or Regular Diffusion Models

**Inputs**: Observation $\boldsymbol{y}$, matrix $H$, score models $\boldsymbol{s}_\theta$, decreasing time steps $\{t_k\}_{k=0}^K$, sampling algorithm from time $t$ to time $s$ using a score function, denoted as $\text{sample}(\text{score}, \boldsymbol{u}_t, t, s)$, and hyperparameter for reconstruction gradient $\lambda_t$.
**Output**: Samples from $p_\theta(\boldsymbol{u})$ where $\boldsymbol{y} = H(\boldsymbol{u})$
Sample $\boldsymbol{u}_K$ from prior distribution $\mathcal{N}(0, \sigma_{t_K}\boldsymbol{I})$
**for** $k \in K, K-1, \cdots 1$ **do**
  # Obtain denoising result from score function $\boldsymbol{s}_\theta(\boldsymbol{u}_k, t_k)$.
  $\hat{\boldsymbol{u}}_0 = \boldsymbol{u}_k + \sigma_{t_k}^2 \boldsymbol{s}_\theta(\boldsymbol{u}_k, t_k)$.
  # Update score based in $\boldsymbol{y}$ and $H$.
  $\tilde{\boldsymbol{s}} = \boldsymbol{s}_\theta(\boldsymbol{u}_k, t_k) + \lambda_t \nabla_{\boldsymbol{u}_k} \|H\hat{\boldsymbol{u}}_0 - \boldsymbol{y}\|_2^2$.
  # Sample based on new score function.
  $\boldsymbol{u}_{k-1} = \text{sample}(\tilde{\boldsymbol{s}}, \boldsymbol{u}_{t_k}, t_k, t_{k-1})$.
**end for**
Return $\boldsymbol{u}_0$

## C.2. Image experiments

**Algorithm 6** Inifnite image generation with `DiffCollage`: training

**Inputs**: Square image data $\mathcal{D}$
**Output**: Shift-invariant score model $\boldsymbol{s}_\theta$ for both factor nodes and variable nodes
**repeat**
  Sample $\boldsymbol{u}_0$ from $\mathcal{D}$
  Random crop $\boldsymbol{u}_0$ by half with $50\%$ probability
  Sample $t$ and Gaussian noise $\epsilon$ with shape of $\boldsymbol{u}_0$
  $\boldsymbol{u}_t = \boldsymbol{u}_0 + \sigma_t \epsilon$
  Gradient descent on $\nabla_\theta[\omega(t) \|\nabla_{\boldsymbol{u}_t} \log q_{0t}(\boldsymbol{u}_t|\boldsymbol{u}_0) - \boldsymbol{s}_\theta(\boldsymbol{u}_t, t)\|^2]$
**util** converged

**Training** To finetune GLIDE [38] on our internal dataset, we first train our base $64 \times 64$ model with a learning rate $1 \times 10^{-4}$ and a batch size 128 for $300K$ iterations. Then we finetune $64 \to 256, 256 \to 1024$ upsamplers for $100K, 50K$ iterations. For the $256 \to 1024$ upsampler, we finetune the upsampler of eDiff-I [3]. Following the prior works [3, 49], we train the $256 \to 1024$ model using random patches of size $256 \times 256$ during training and apply it on $1024 \times 1024$ resolution during inference. We utilize AdamW optimzer [33] and apply exponential moving average (EMA) with a rate $0.999$ during training. The base $64 \times 64$ diffusion model is trained to be conditioned on image CLIP embeddings with a random drop rate $50\%$ while the two upsampling diffusion models are only conditioned on low-resolution images. For the diffusion model conditioned on semantic segmentation maps, we replace the first layer of our pre-trained base $64 \times 64$ model and concatenate embeddings of semantic segmentation maps and noised image inputs. We further finetune the diffusion model for another $100K$ iterations conditioned on segmentation.

For experiments on LHQ [53] and LSUN [67] Tower, we train diffusion models from scratch with the U-net architecture proposed in Dhariwal *et al*. [13]. Thanks to its success in LSUN and ImageNet [12], we adopt its hyperparameters for LSUN dataset in [13, Table 11]. Due to limited computational resources, we train diffusion models with channel size 192 and batch size 128 for $100K$ iterations instead of the recommended hyperparameters. We follow the data preprocessing in Skorokhodov *et al*. [53, Algorithm 1] with its official implementation [2], which extracts a subset with approximately horizontally invariant statistics from original datasets.

Thanks to the shift-invariant property of infinite images, we use the same diffusion model to fit both factor and variable nodes, where the width of images over the variable node is half of the width of factor nodes. The dataset for variable nodes consists of random cropped images from factor nodes. We list its training algorithm in Algorithm 6. We apply a similar strategy to train segmentation-conditioned diffusion models. We adopt VESDE and preconditioners proposed in Karras *et al*. [25] to train our diffusion models.

[2] https://gist.github.com/universome/3140f74058a48aa56a556b0d9e24e857

| Method | R Precision (top 3)↑ | FID↓ | Multimodal Dist↓ | Diversity→ |
|---|---|---|---|---|
| Real data | 0.798±0.002 | 0.001±0.000 | 2.960±0.006 | 9.471±0.100 |
| MDM [59] | 0.605±0.005 | 0.492±0.036 | 5.607±0.028 | 9.383±0.070 |
| Baseline | 0.298± 0.006 | 10.690 ±0.179 | 7.512±0.039 | 6.764±0.069 |
| Replacement | 0.567± 0.008 | 1.281 ±0.177 | 5.751±0.034 | 9.184±0.122 |
| Reconstruction | 0.585± 0.007 | 1.012 ±0.080 | 5.716±0.033 | 9.175±0.120 |
| DiffCollage | 0.611± 0.004 | 0.605 ±0.082 | 5.569±0.017 | 9.372±0.109 |

Table 4. Performance on every metric is reported based on a mean and standard derivation of 20 independent evaluations.

**Sampling** Regarding sampling image diffusion models, we use the stochastic sampler in Karras *et al*. [25] with 80 sampling steps and default hyperparameters. We find stochastic samplers are slightly better than deterministic samplers in DiffCollage. For quantitative comparison on our internal dataset, we have the same CLIP embedding for both factor and variable nodes in one graph while we use unconditional generation on LHQ and LSUN Tower. We use the same sampler for baseline and autoregressive methods based on replacement or reconstruction. To connect different styles and real images with a linear chain graph, we interpolate conditional signals with spherical linear interpolation [62]. We find DiffCollage with Algorithm 4 can produce satisfying samples for conditional generation efficiently. More visual examples are included in Appendix E.

### C.3. Motion experiments

We use the pre-trained diffusion model[3] from [59] and only make the following modifications during sampling.

- Similar to experiments in images, we inpaint motion sequences by masking $50\%$ content in the sliding window for Replacement and Reconstruction methods.
- All experiments employ the deterministic DDIM sampler [55] with 50 steps.
- We use the same prompt to denoise both factor and variables nodes for long motion experiments benchmark experiments results and Tab. 4.
- To composite motions with multiple actions, we decompose the given long prompts into several short sentences manually so that each sentence only consists of one or two actions similar to prompts in the training data. Then we assign each factor $\boldsymbol{y}[f_j]$ with one short prompt sequentially and unconditional null token for the variables node.
- Analogous to circle image generation, we add a factor node connected to the head and tail variable nodes in the factor graph.

We include standard derivation for long motion experiments in Tab. 4.

## D. Limitations

Despite the clear advantages that DiffCollage has over traditional methods, DiffCollage is no silver bullet for every large content generation problem. We discuss some limitations below.

**Conditional independence assumptions.** Since we use diffusion models trained on smaller pieces of the content, DiffCollage place conditional independence assumptions over the joint distribution of the large content, similar to autoregressive outpainting methods. Sometimes this assumption is reasonable (such as long images for landscape or "corgis having dinner at a long table"), but there are cases where the long-range dependency is necessary for generating the content. For example, generating a long image of a snake would be difficult with DiffCollage, since we drop the conditional dependencies between the head and the tail of the snake, and it is possible that our snake would have two heads or two tails. Part of this can be mitigated by providing global conditioning information, such as the segmentation maps in landscapes.

[3] https://github.com/GuyTevet/motion-diffusion-model

**Memory footprint.** We reduce the latency of the long content generation by running the diffusion model computations in parallel, and it comes at a cost of using more peak memory than autoregressive methods.

**Number of steps in the sampler.** To ensure global consistency, information needs to flow through the factor graph. This is done by the sum over the overlapping regions in each iteration, so it can be treated as some kind of "message passing" behavior. Similar to "message passing", many iterations may be needed if the graph diameter is large (even when some global conditioning information is given). For example, for a linear chain with length $L$, we may need the sampler to run $O(L)$ times to get optimal results. Empirically we also find sampling with our method using very few steps in generating infinite images, such as $35$, may result in artifacts. However, we note that this is still much better than the autoregressive counterpart; for a `DiffCollage` implementation that requires $O(L)$ steps of iteration, the reconstruction/replacement methods would require $O(L \times K)$ steps, where $K$ is the number of iterations for the small diffusion models.

## E. Additional samples

We include more high-quality samples and motion videos in our supplementary materials.

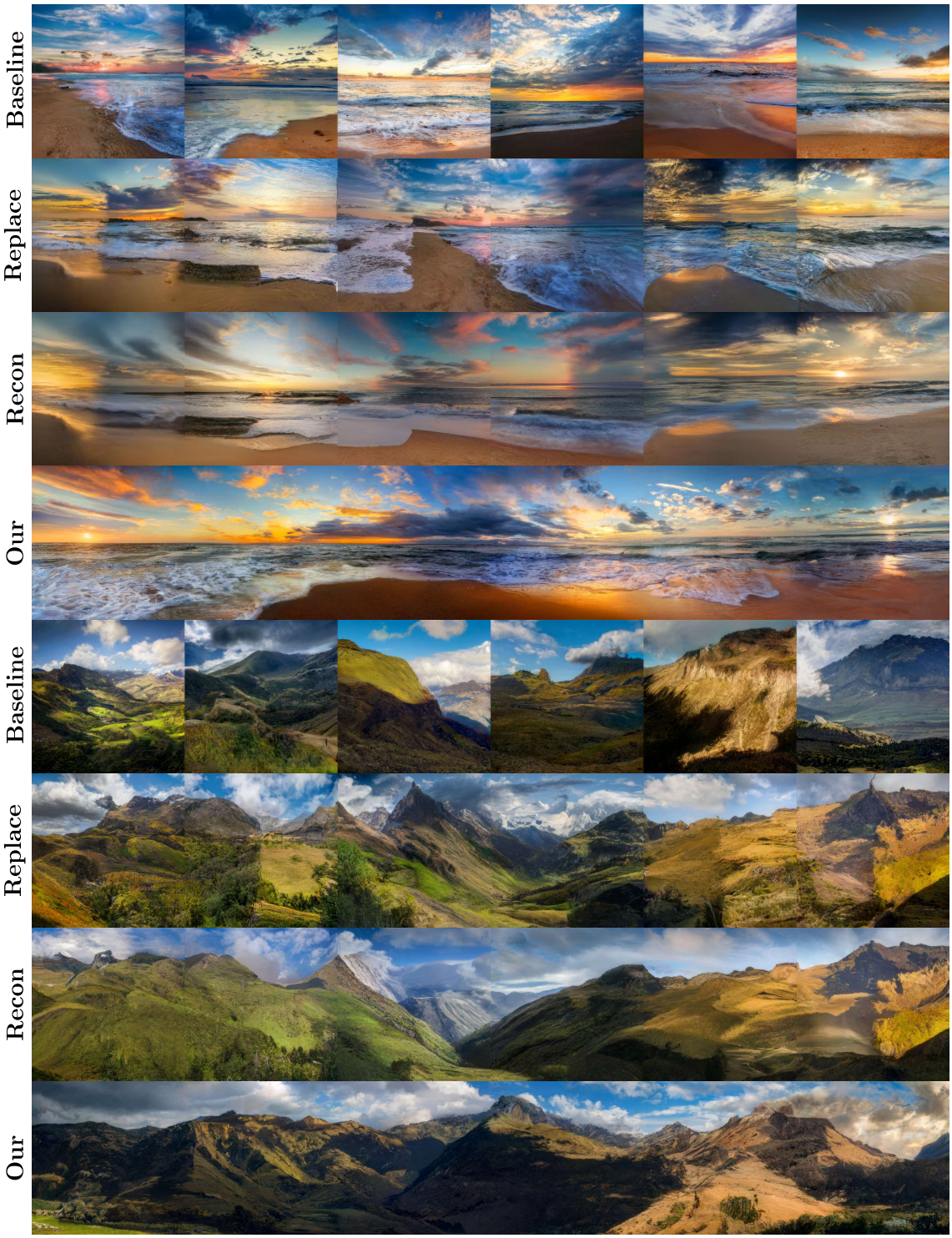


Figure 10. More comparison.


Figure 11. Inpainting on non-square images. The diffusion models based on smaller patches are run in parallel.


Figure 12. Inpainting on non-square images. The diffusion models based on smaller patches are run in parallel.

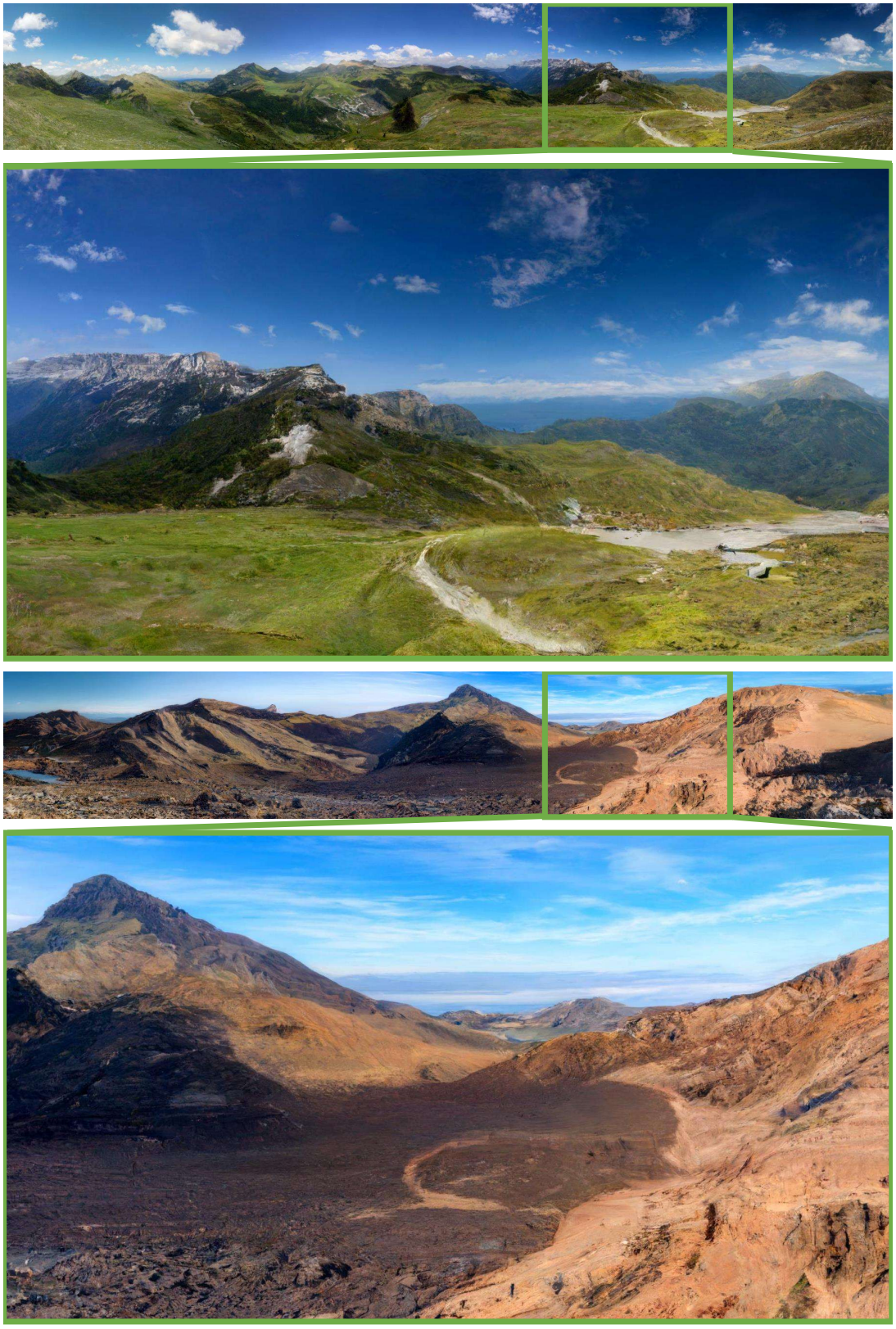

Figure 13. DiffCollage on generating long landscape images. Parts are being zoomed in for high-resolution details.

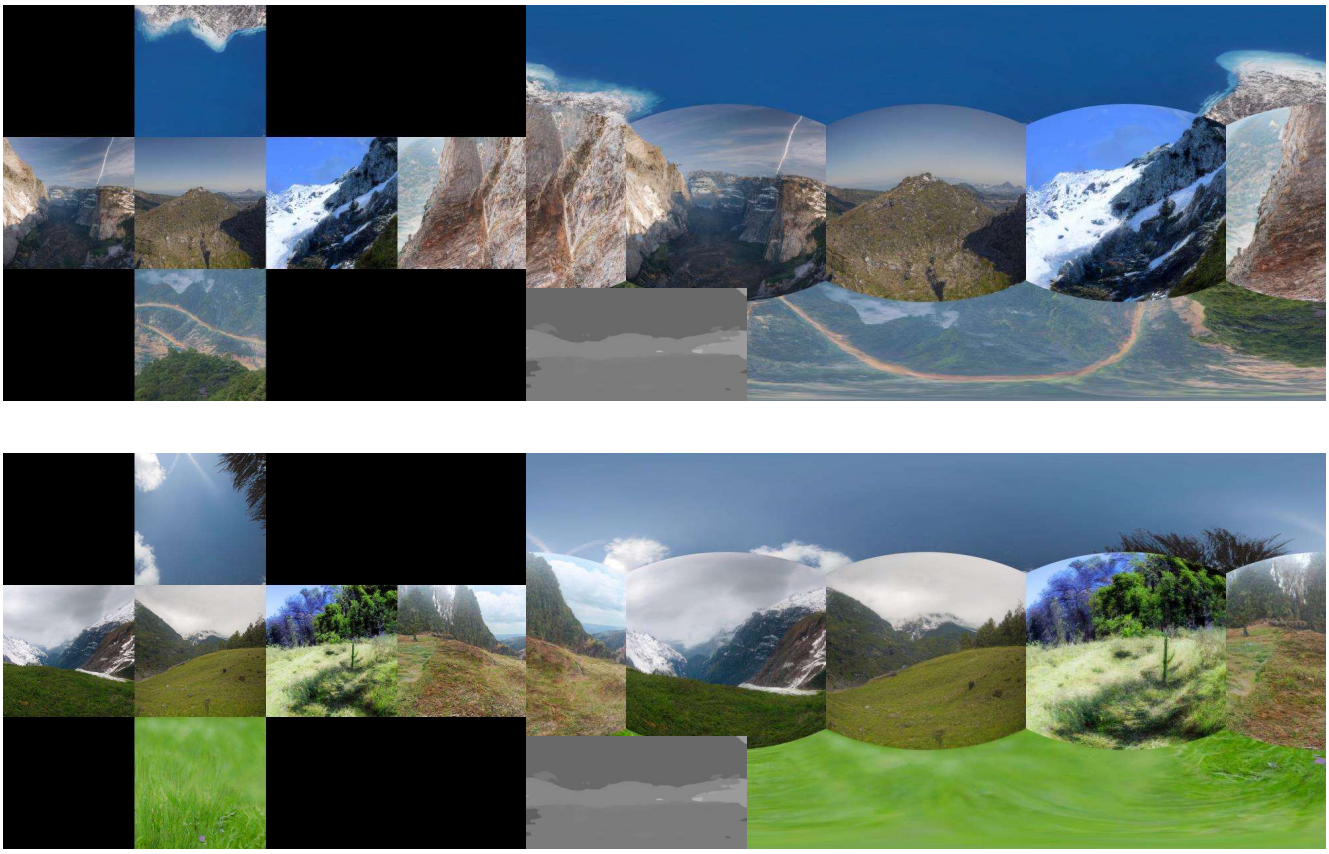

Figure 14. Baseline method on 360 cubemap image. (Left) Cubemap representation; (Right) Equirectangular representation; (Lower middle) Semantic segmentation map used to condition the model. Even with a globally-consistent semantic segmentation map, individually processing the patches will lead to the cube faces being quite inconsistent with one another.

Figure 15. DiffCollage on 360 cubemap image. (Left) Cubemap representation; (Right) Equirectangular representation; (Lower middle) Semantic segmentation map used to condition the model. DiffCollage is able to "connect" the different faces and produce a globally consistent 360 degree image.

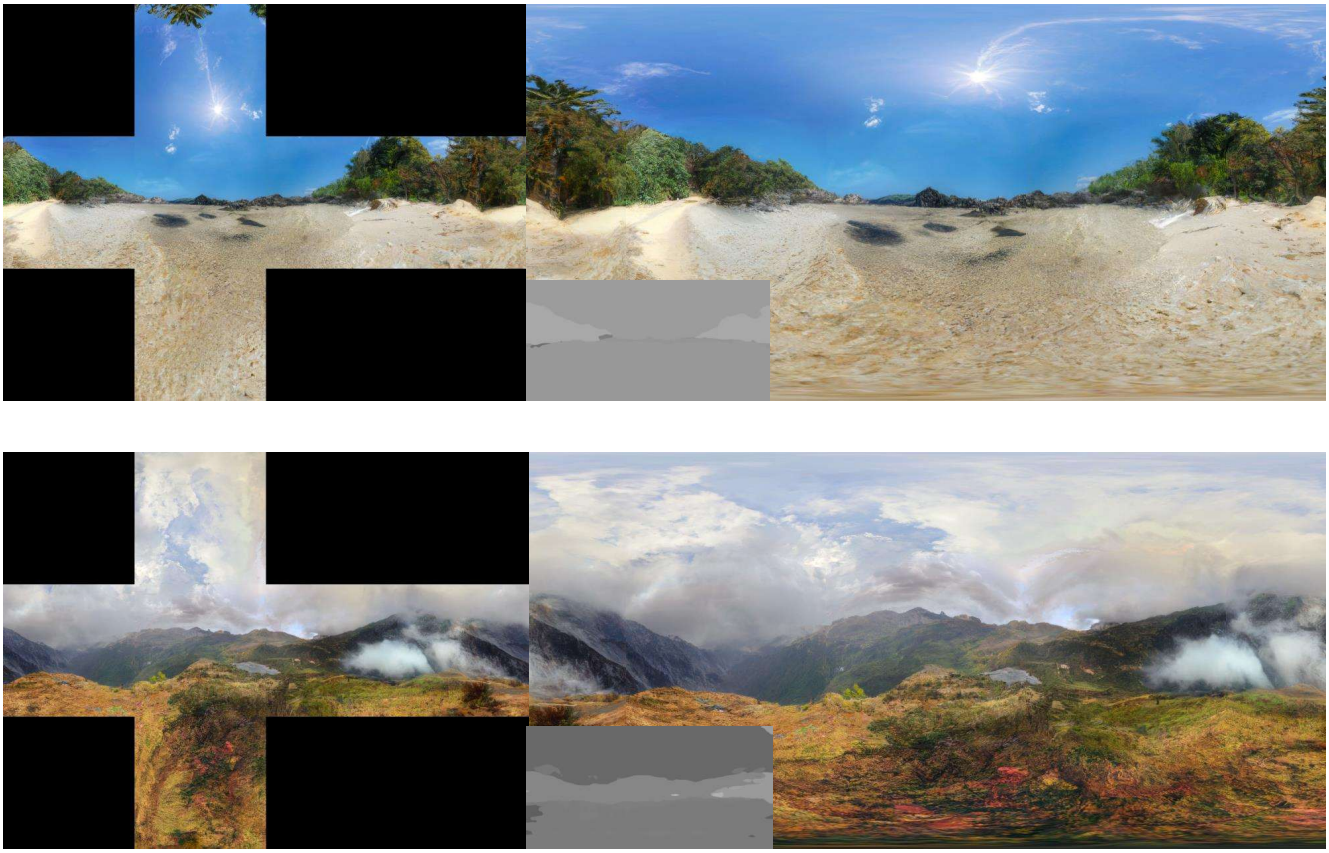

Figure 16. `DiffCollage` on 360 cubemap image. (Left) Cubemap representation; (Right) Equirectangular representation; (Lower middle) Semantic segmentation map used to condition the model.