A. Discussion

A.1. Limitations

Our method has the following limitations:

1. Shift-equivariance of our generator is limited only to the step size equal to the size of a patch, i.e., one cannot take a step smaller than the patch size. In our case, it equals 16 pixels for 256^2-resolution generator and 64 pixels for 1024^2-resolution one. In this way, our generator is *periodic* shift-equivariant [55].

2. As shown in 9, connecting too different scenes leads to seaming artifacts. There are two ways to alleviate this: sampling from the same distribution mode (after clustering the latent space, like in Figure 25) and increasing the distance between the anchors. The latter, however, leads to repetitions artifacts, like in 10.

3. As highlighted in 3.3, infinite image generation makes sense only for datasets of images with spatially invariant statistics and our method exploits this assumption. In this way, if one does not preprocess LSUN Bedroom with Algorithm 1, then the performance of our method will degrade by a larger margin compared to TT [10] for *non-infinite* generation. However, after preprocessing the dataset, our method can learn to connect those "unconnectable" scenes (see Figure 12). This limitation *will not* be an issue in practice because when one is interested in non-infinite image generation, then a non-infinite image generator is employed. And when on tries to learn infinite generation on a completely "unconnectable" dataset, then any method will fail to do so (unless some external knowledge is employed to rule out the discrepancies).

Figure 12: Connecting the "unconnectable" scenes of LSUN Bedroom. Though LSUN Bedroom is a dataset of images with spatially non-invariant statistics (see Appendix C), our method is still able to reasonably connect its scenes after the dataset is preprocessed with Algorithm 1.

4. Preprocessing the dataset with our proposed procedure in Section 3.3 changes the underlying data distribution since it filters away images with spatially non-invariant statistics. Note that we used the same processed datasets for our baselines in all the experiments to make the comparison fair.

5. Our proposed ∞-FID metric does not capture possible stitching problems. To alleviate this, we additionally propose *spatial perceptual path length* (SPPL) to catch too fast content changes: "walk" through the infinite image with the step size of K pixels and compute the average perceptual distance between the current frame and the frame K pixels to the right. We report it on Figure 27.

A.2. Questions and Answers

Q: *What is spatial equivariance? Why is it useful? Is it unique to your approach, or other image generators also have it?*

A: We call a decoder *spatially-equivariant* (shift-equivariant) if shifting its input results in an equal shift of the output. It is not to be confused with spatial *invariance*, where shifting an input does not change the output. The equivariance property is interesting because it is a natural property that one seeks in a generator, i.e., by design, we want a decoder to move in accordance with the coordinate space movements. It was also shown that shift-equivariance improves the performance

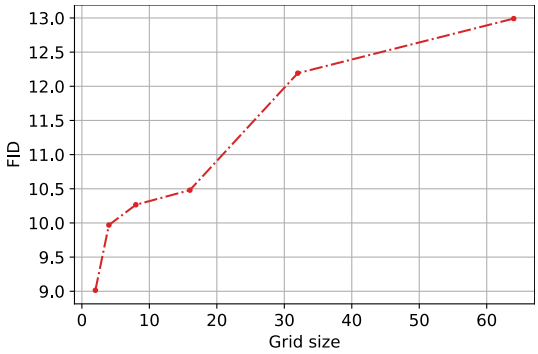


Figure 13: Ablating the grid size of the ALIS generator on LHQ 256^2 for patchwise generation. Reducing the grid size improves the performance, but in the expense of not being able to do small steps in the coordinate space at test time. I.e. with 2×2 grid, one would achieve the highest FID score but will be able to generate an image only by large chunks (of size $W/2$) which might be undesirable.

Table 2: Comparison of existing infinite image generators. We used LocoGAN [57] as our benchmark since it shares all the features of previously developed infinite-texture image generators and was additionally tested on non-texture datasets. N/A denotes "not applicable": while ∞-GAN was trained on non-texture images, its generations are in the "texture-like" spirit.

	SpatialGAN [20]	PS-GAN [2]	∞-GAN [29]	LocoGAN [57]	LocoGAN [57]+SG2 [57]+Fourier [42, 45]
Has local latents?	✓	✓	✓	✓	✓
Has global latents?	✓	✓	✓	✓	✓
Has periodic positional embeddings?		✓	✓	✓	✓
Not limited to a single image?	✓	✓		✓	✓
Was employed for non-texture datasets?			N/A	✓	✓
Is big?					✓

[55] of both decoder and encoder modules. Traditional generators do not have it due to upsampling [55]. While we have upsampling procedures too, in our case they do not break *periodic* shift-equivariance because they are patchwise.

Q: *Why did you choose the grid size of 16?*

A: The grid size of 16 allows to perform small enough steps in the coordinate space while not losing the performance too much (as also confirmed by [25]). Small steps in the coordinate space are needed from a purely design perspective to generate an image in smaller portions. In the ideal case, one would like to generate an image pixel-by-pixel, but this extreme scenario of having 1×1 patches worsens image quality [43]. We provide the ablation study for the grid size in Figure 13.

Q: *Why didn't you compare against other methods for infinite image generation?*

A: There are two reasons for it. First, to the best of our knowledge, there are no infinite generators of *semantically complex* images, like LSUN Tower or nature landscapes — only texture-like images. This is why we picked "any-aspect-ratio" image generators as our baselines: LocoGAN [57] and Taming Transformers [10]. Second, our LocoGAN+SG2+Fourier baseline encompasses the previous developments in infinite texture synthesis (like [20, 2, 29]). It has the same architecture as the state-of-the-art PS-GAN [2], but is tested not only on textures, but also semantically complex datasets, like LSUN Bedrooms and FFHQ. See Table 2 for the benchmarks comparison in terms of features.

Q: *How is your method different from panoramas generation?*

A: A panorama is produced by panning, i.e. camera rotation, while our method performs *tracking*, i.e. camera transition. Also, a (classical) panorama is limited to 360° and will start repeating itself in the same manner as LocoGAN does in Figure 7.

Q: *How is your method different from image stitching?*

A: Image stitching connects nearby images of the *same* scene into a single large image, while our method merges *entirely* different scenes.

Q: *Does your framework work for 2D (i.e., joint vertical + horizontal) infinite generation? Why didn't you explore it in the paper?*

A: Yes, it can be easily generalized to this setup by positioning anchors not on a 1D line (as shown in Figure 2), but on

a 2D grid instead and using bilinear interpolation instead of the linear one. The problem with 2D infinite generation is that there are no *complex* datasets for this, only pattern-like ones like textures or satellite imagery, where the global context is *unlikely* to span several frames. And the existing methods (e.g., LocoGAN [57], ∞-GAN [29], PS-GAN [2], SpatialGAN [20], TileGAN [11], etc.) already mastered this setup to a good extent.

Q: *How do you perform instance normalization for layers with* 16×16 *resolution? Your patch size would be equal to* 1×1*, so* $\boldsymbol{\sigma}(\boldsymbol{x}) = NaN$ *(or* 0*) for it.*

A: In contrast to CocoGAN[25], we use *vertical* patches (see Appendix B), so the minimal size of our patch is 16×1, hence the std is defined properly.

Q: *Why* ∞*-FID for LocoGAN is so high?*

A: LocoGAN generates a repeated scene and that's why it gets penalized for mode collapse.

B. Implementation details

B.1. Architecture and training

We preprocess all the datasets with the procedure described in Algorithm 1 with a threshold parameter of 0.7. We train all the methods on identical datasets to make the comparison fair.

As being said in the main paper, we develop our approach on top of the StyleGAN2 model. For this, we took the official StyleGAN2-ADA implementation and disabled differentiable augmentations [22, 56] since Taming Transformers [10] do not employ such kind of augmentations, so it would make the comparison unfair. We used a small version of the model (config-e from [24]) for our 256^2 experiments and the large one (config-f from [24]) for 1024^2 experiments. They differ in the number of channels in high-resolution layers: the config-f-model uses twice as big dimensionalities for them.

We preserve all the training hyperparameters the same. Specifically, we use $\gamma = 10$ for R1-regularization [31] for all StyleGAN2-based experiments. We use the same probability of 0.9 for style mixing. In our case, since on each iteration we input 3 noise vectors $\boldsymbol{w}_l, \boldsymbol{w}_c, \boldsymbol{w}_r$ we perform style mixing on them independently. Following StyleGAN2, we also apply the Perceptual Path Loss [24] regularization with the weight of 2 for our generator G. We also use the Adam optimizer to train the modules with the learning rates of 0.0025 for G and D and betas of 0.0 and 0.99.

As being said in the main paper, CoordConst and CoordConv3x3 are analogs of StyleGAN's Const block and Conv3x3 respectively, but with coordinates positional embeddings concatenated to the hidden representations. We illustrate this in Figure 16. As being said in Section 3, our Conv3x3 is not modulated because one cannot make the convolutional weights be conditioned on a position efficiently and we use SA-AdaIN instead to input the latent code. Const block of StyleGAN2 is just a learnable 3D tensor of size $512 \times 4 \times 4$ which is passed as the starting input to the convolution decoder. For CoordConst layer, we used padding mode of repeat for its constant part to make it periodic and not to break the equivariance.

To avoid numerical issues, we sampled δ only at discrete values of the interval $[0, 2d - W]$, corresponding to a left border of each patch.

Each StyleGAN2-based model was trained for 2500 kimgs (i.e. until D saw 25M real images) with a batch size of 64 (distributed with individual batch size of 16 on 4 GPUs). Training cumulatively took 2.5 days on 4 V100 GPUs.

For Taming Transformer, we trained the model for 5 days in total: 2.5 days on 4 V100 GPUs for the VQGAN component and 2.5 days on 4 V100 GPUs for the Transformer part. VQGAN was trained with the cumulative batch size of 12 and the Transformer part — with 8. But note that in total it was consuming *more* GPU memory compared to our model: 10GB vs 6GB per a GPU card. We used the official implementation with the official hyperparameters[1].

B.2. Patchwise generation

As being said in Section 3, following the recent advances in coordinate-based generators [25, 1, 43], our method generates images via spatially independent patches, which is illustrated in Figure 15c. The motivation of it is to force the generator learn how to stitch nearby patches based on their positional information and latent codes. This solved the "padding" problem of infinite image generation: since traditional decoders extensively use padding in their implementations, this creates stitching artifacts when the produced frames are merged naively, because padded values are actually filled with neighboring context during the frame-by-frame infinite generation (see Figure 15a). A traditional way to solve this [20, 2, 57, 29] is to merge the produced frames with overlaps, like depicted in Figure 15b.

[1] https://github.com/CompVis/taming-transformers

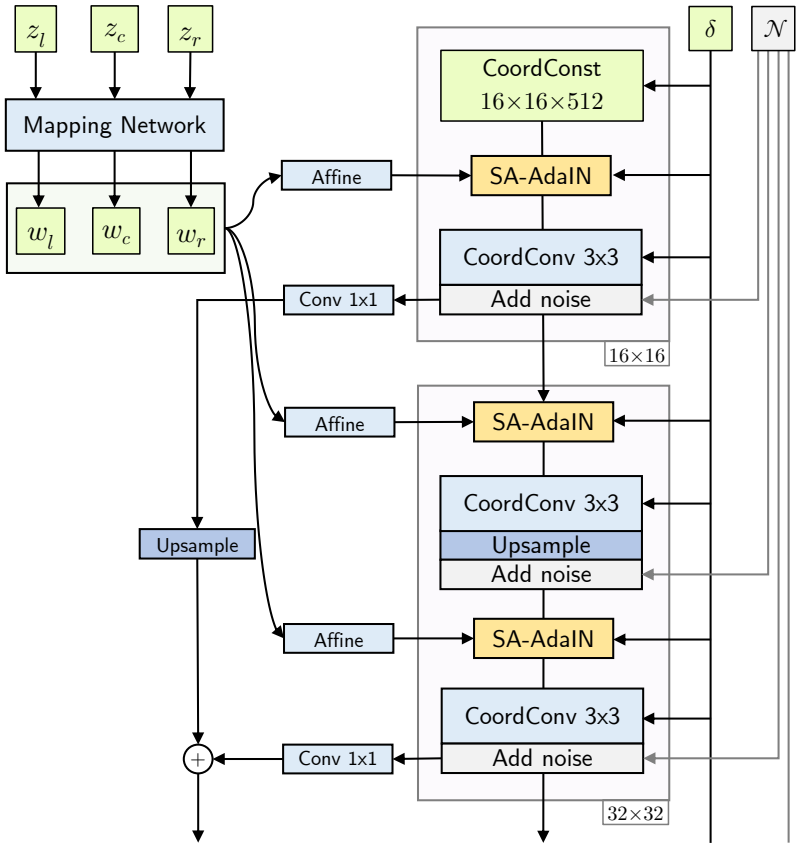
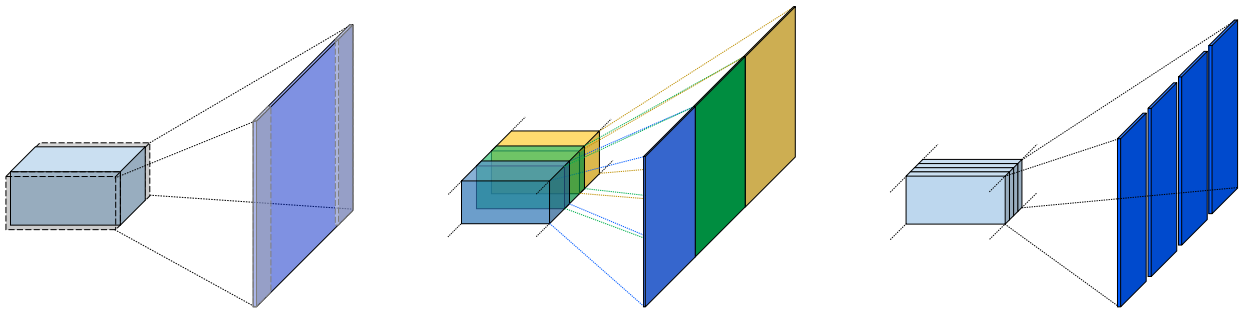


Figure 14: Full architecture of our G.

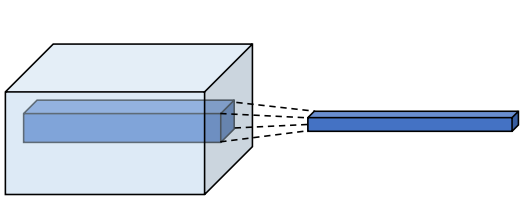


(a) Padding problem (b) Overlapped generation [20, 57, 2, 29, 10] (c) Patchwise generation [25, 43, 1]

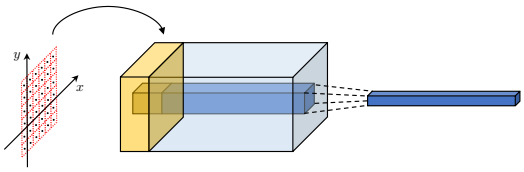
Figure 15: (a) A padding problem that occurs with infinite image generation; (b, c) two strategies to alleviate it. For ALIS, we use patchwise generation and show how to incorporate spatially varying *global* latent codes into it.

C. Which datasets are connectable?

In this section, we elaborate on the importance of having spatially invariant statistics for training an infinite image generator. Since we consider only horizontal infinite image generation, we are concerned about horizontally invariant statistics, but the idea can be easily generalized to vertical+horizontal generation. As noted by [10], to train a successful "any-aspect-ratio" image generator from a dataset of just independent frames, there should be images containing not only scenes, but also transitions between scenes. We illustrate the issue in Figure 17. And this property does not hold for many computer vision datasets, for example FFHQ, CelebA, ImageNet, LSUN Bedroom, etc. To test if a dataset contains such images or not and to



(a) Conv3x3



(b) CoordConv3x3. Orange part of the hidden representation denotes coordinates embeddings.

Figure 16: Illustration of Conv3x3 and CoordConv3x3. For CoordConv3x3 there are different strategies on how to compute coordinates embeddings, for ALIS, we use periodic positional encoding [42, 45].

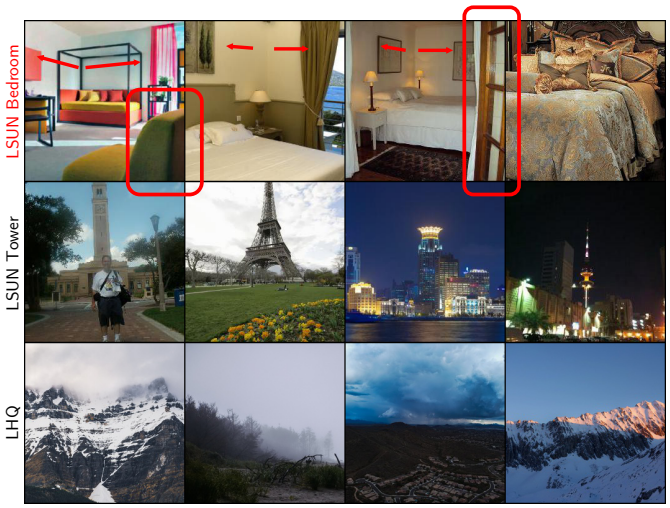


Figure 17: Random samples from LSUN Bedroom (top), LSUN Tower (middle) and LHQ (bottom). While LSUN Tower and LHQ have (approximately) horizontally invariant statistics, for LSUN Bedroom this does not hold, which is highlighted in red. For LSUN Bedroom, the first problem occurs due to walls: it is impossible to continue a sample to the left due to wall hitting the camera. And there are no samples in LSUN Bedroom which would have the transition between those walls directions. The second problem is in too-close-to-camera objects. We propose Algorithm 1 to test if the dataset is connectable and to find a connectable subset of it. After preprocessing LSUN Bedroom, our algorithm can fit it, as shown in Figure 12. The importance of this procedure is also confirmed by FID scores in Table 3.

extract a subset of good images, we propose the following procedure.

Given a dataset, train a binary classifier to predict whether a given half of an image is the left one or right one. Then, if the classifier has low confidence on the test set, the dataset contains images with spatially invariant statistics, since image left/right patch locations cannot be confidently predicted. Now, to extract the subset of such good images, we just measure the confidence of a classifier and pick images with low confidence, as described in Algorithm 1.

To highlight the importance of this procedure, we trained our model on different subsets of LHQ and LSUN Tower, varying the confidence threshold t. As one can see from Table 3, our procedure is essential to obtain decent results. Also the confidence threshold should not be too small, because too few images are left making the training unstable.

As a classifier, we select an ImageNet-pretrained WideResnet50 and train it for 10 epochs with Adam optimizer and learning rates of 1e-5 for the body and 1e-4 for the head. We consider only square center crops of each image.

Also, we tested our "horizontal invariance score" for different popular datasets and provide them in Table 4. As one can see from this table, our proposed dataset contains images with much more spatially invariant statistics.

We used $t = 0.95$ for LHQ and $t = 0.7$ for LSUN datasets.

Algorithm 1: Extract a subset with (approximately) horizontally invariant statistics.

Input : Dataset D of $(H \times W \times C)$-sized images
Input : Confidence threshold t.
Output: Dataset $D_t \subseteq D$ of horizontally invariant images.
Initialize a binary classifier C_θ;
Split D into subsets $D_{\text{train}}, D_{\text{val}}, D_{\text{rest}}$ with ratio 2:1:7;
while *not converged* **do**
  Sample a training batch $\boldsymbol{X} \subseteq D_{\text{train}}$;
  for $i \leftarrow 1$ **to** $|\boldsymbol{X}|$ **do**
    Randomly sample side $s^{(i)} \in \{\text{left}, \text{right}\}$;
    if $s^{(i)}$ = left **then**
      $x^{(i)} \leftarrow x^{(i)}[:, : W/2]$; // Select the left half
      $y^{(i)} \leftarrow 0$;
    else
      $x^{(i)} \leftarrow x^{(i)}[:, W/2 :]$; // Select the right half
      $y^{(i)} \leftarrow 1$;
    end
  end
  Update C_θ based on training batch $\{x^{(i)}, y^{(i)}\}_{i=1}^{|\boldsymbol{X}|}$;
end
Evaluate C_θ on D_{rest} to obtain scores for each image half $\boldsymbol{R} = (r_{\text{left}}^{(1)}, r_{\text{right}}^{(1)}, r_{\text{left}}^{(2)}, ..., r_{\text{left}}^{(N)}, r_{\text{right}}^{(N)})$;
Set $D_t \leftarrow \{x^{(i)} | x^{(i)} \in D_{\text{rest}} \wedge \max\{r_{\text{left}}^{(i)}, r_{\text{right}}^{(i)}\} < t\}$;
Return D_t;

Table 3: FID scores of ALIS for different confidence thresholds t in Algorithm 1 on LHQ 256^2 and LSUN Tower 256^2. Preprocessing the dataset with the proposed procedure is essential to obtain decent results, since it filters out "bad" images, i.e. images with spatially non-invariant statistics. We select $t = 0.7$ for LSUN datasets and $t = 0.95$ for LHQ since decreasing t filters away too many images which also hurts the FID score. "Dataset size" is the number of images remaining after the filtering procedure. For $t = 1$, the dataset is untouched.

Method	LSUN Tower 256^2		LHQ 256^2	
	FID	Dataset size	FID	Dataset size
$t = 1$	17.51	708k	23.23	90k
$t = 0.99$	8.68	168k	10.18	50k
$t = 0.95$	8.85	116k	10.48	38k
$t = 0.7$	8.83	59k	11.49	19k
$t = 0.5$	9.73	41.5k	14.55	13k

D. Landscapes HQ dataset

To collect the dataset we used two sources: Unsplash[2] and Flickr[3]. For the both sources, the acquisition procedure consisted on 3 stages: constructing search queries, downloading the data, refining the search queries based on manually inspecting the subsets corresponding to different search queries and manually constructed blacklist of keywords.

After a more-or-less refined dataset collection was obtained, a pretrained Mask R-CNN model was employed to filter out the pictures which likely contained an object. The confidence threshold for "objectivity" was set to 0.2.

For Unsplash, we used their publicly released dataset of photos descriptions[4] to extract the download urls. It was preprocessed by first searching the appropriate keywords with a whitelist of keywords and then filtering out images with a blacklist

[2] https://unsplash.com/
[3] https://www.flickr.com/
[4] https://unsplash.com/data

Table 4: Spatial invariance scores for different datasets. We used Algorithm 1 to train a classifier and computed its mean accuracy on D_{rest}. In all the cases, we used center crops.

Dataset	Mean Test Confidence
LSUN Bedroom	95.2%
LSUN Tower	92.7%
LSUN Bridge	88.6%
LSUN Church	96.1%
FFHQ	99.9%
ImageNet	99.8%
LHQ	82.5%

of keywords. A whitelist of keywords was constructed the following way. First, we collected 230 prefixes, consisting on geographic and nature-related places. Then, we collected 130 nature-related and object-related suffixes. A whitelist keyword was constructed as a concatenate of a prefix and a suffix, i.e. $230 \times 130 = 30k$ whitelist keywords. Then, we extracted the images and for each image enumerated all its keywords and if there was a keyword from our blacklist, then the image was removed. A blacklist of keywords was constructed by trial and error by progressively adding the keywords from the following categories: animals, humans, human body parts, human activities, eating, cities, buildings, country-specific sights, plants, drones images (since they do not fit our task), human belongings, improper camera positions. In total, it included 300 keywords. We attach both the whitelist and the blacklist that were used in the supplementary. In total, 230k images was collected and downloaded with this method.

For Flickr, we constructed a set of 80 whitelist keywords and downloaded images using its official API[5]. They were much more strict, because there is no way to use a blacklist. In total, 170k images were collected with it.

After Unsplash and Flickr images were downloaded, we ran a pretrained Mask R-CNN model and removed those photos, for which objectivity confidence score was higher than 0.2. In total, that left 60k images for Unsplash and 30k images for Flickr.

We found, that even all the above procedures were not enough to eliminate all the noise, found in the data, downloaded from the internet. So we manually filtered out a subset of 10k images which are guaranteed not to contain any noise and represent high-quality landscape pictures. This constitutes "LHQ-base": a manually constructed subset of LHQ with high-quality images.

The images come in one of the following licenses: Unsplash License, Creative Commons BY 2.0, Creative Commons BY-NC 2.0, Public Domain Mark 1.0, Public Domain CC0 1.0, or U.S. Government Works license. All these licenes allow the use of the dataset for research purposes.

We depict 100 random samples from LHQ in Figure 18 and a word map of keywords in Figure 19.

E. Remarks on comparison to Taming Transformers

Using Taming Transformers (TT) [10] as a baseline is complicated for the following reasons:

- The original paper was mainly focused on conditional image generation. Since our setup is unconditional (and thus harder), the produced samples are of lower quality and it might confuse a reader when comparing them to conditionally generated samples from the original TT's paper [10].
- It is not a GAN-based model (though there is some auxiliary adversarial loss added during training) — thus, in contrast to LocoGAN or ALIS, it cannot be reimplemented in the StyleGAN2's framework. This makes the comparison to StyleGAN2-based models harder since StyleGAN2 is so well-tuned "out-of-the-box". However, it must be noted that TT is a very recent method and thus was developed with access to all the recent advances in generative modeling. As being said in Section 5, we used the official implementation with the official hyperparameters for unconditional generation training. In total, it was trained for twice as long compared to the rest of the models: 2.5 days on 4 V100 GPUs for the VQGAN part and 2.5 days on 4 V100 GPUs for the Transformer part.
- Autoregressivy inference makes it very slow at test time. For example, it took 8 days to compute FID and ∞-FID scores on a single V100 GPU for a single experiment.

[5] https://www.flickr.com/services/api/

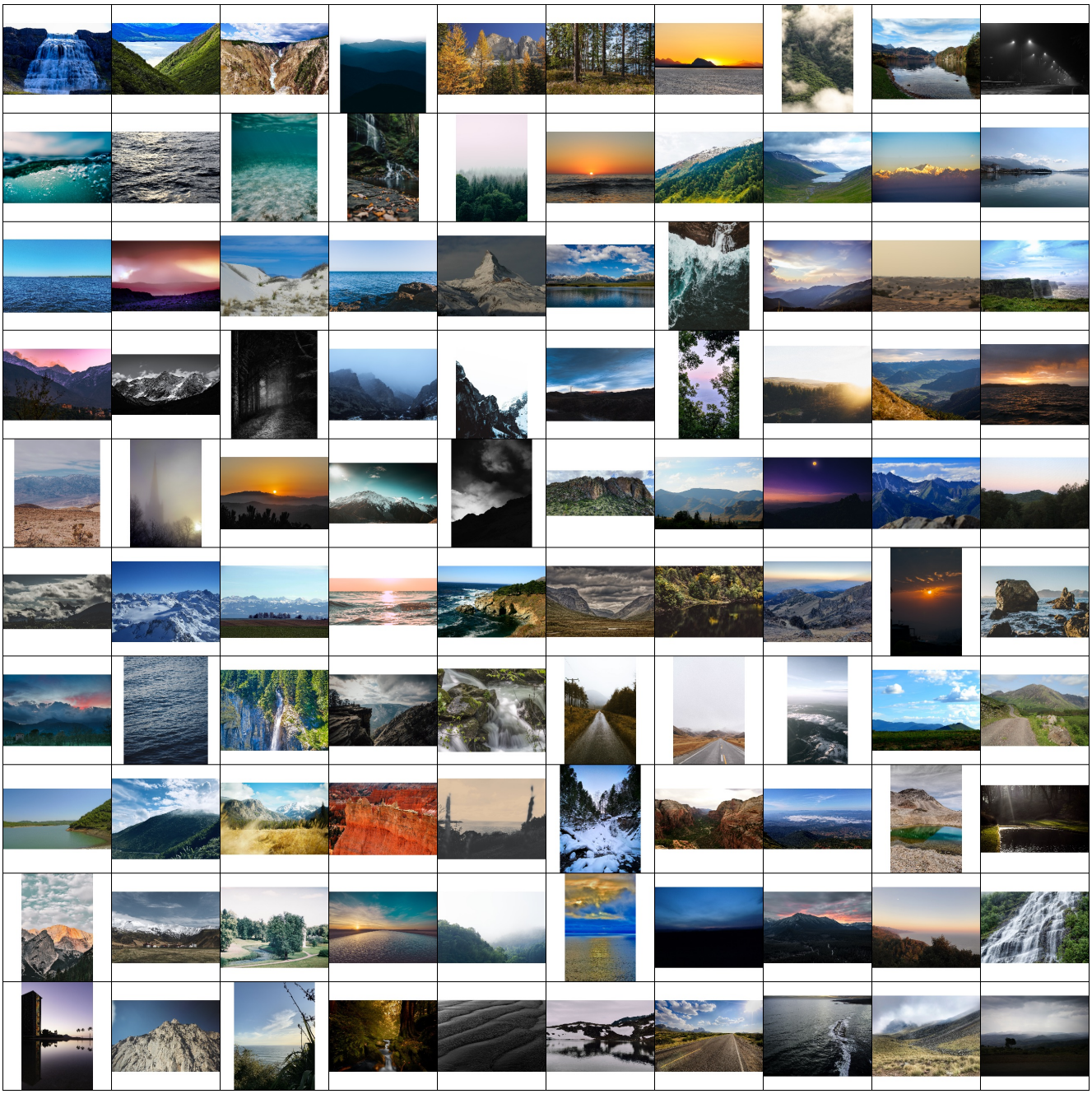


Figure 18: 100 random images from LHQ dataset, that we introduce in our work. It is a diverse dataset of very different scenes and in high resolution $\geq 1024^2$. We downsized the images for this figure to avoid performance issues.

- Taming Transformer's decoder uses GroupNorm layer [52]. GroupNorm, as opposed to other normalization layers (like BatchNorm [19]) does not collect running statistics and computes from the hidden representation even at test-time. It becomes problematic when generating an "infinite" image frame by frame, because neighboring frames use different statistics at forward pass. This is illustrated in Figure 20. To solve the for ∞-FID, instead of generating a long $256 \times (50000 \cdot 256)$ image, we generated 500 $256 \times (100 \cdot 256)$ images. Note that it makes the task easier (and improves the score) because this increases the diversity of samples, which FID is very sensitive to.

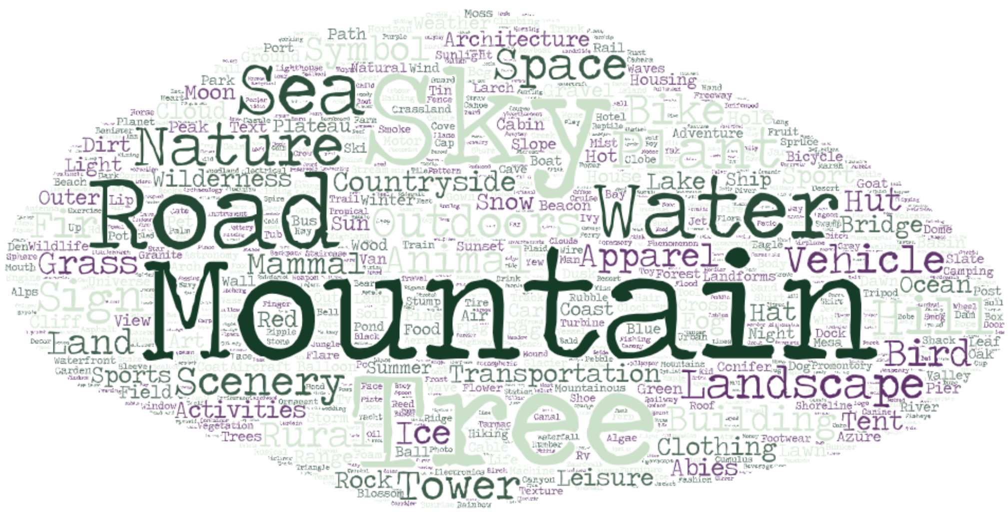


Figure 19: A word map of keywords in the LHQ dataset.

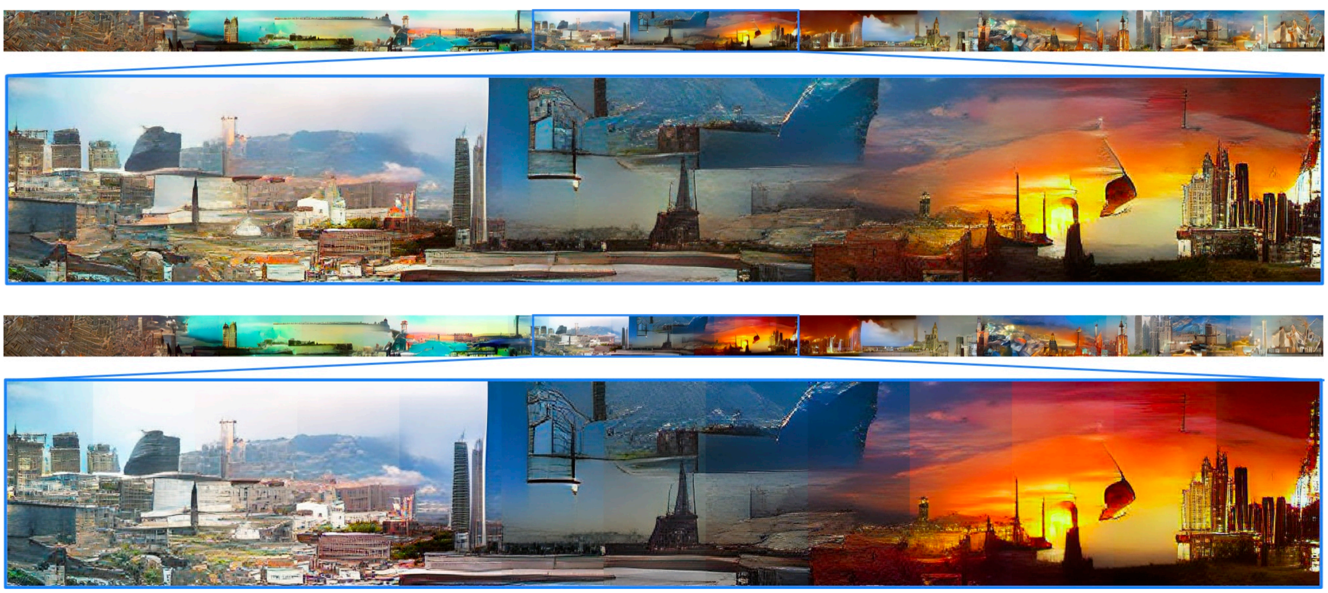


Figure 20: Illustration of GroupNorm artifacts of TT [10] when frame-by-frame infinite generation is considered. Top: normal samples, bottom: overlapped generation (see Figure 15). Since TT's decoder uses GroupNorm [52], it computes activations statistics across the whole image, i.e. it needs the whole image to be available simultaneously during inference. If frame-by-frame generation is employed at test-time (to produce an infinite image, for example), then it leads to the illustrated artifacts.

F. Additional samples

G. Spatial Perceptual Path Length

Our proposed ∞-FID does not capture stitching problems. To alleviate this, we propose spatial perceptual path length (SPPL) metric, which is computed the following way. First, sample an "infinite" (i.e. very long image). Then, walk over it frame by frame with a step size of K pixels and compute the perceptual distance between each pair of neighbouring frames. Finally, report the average distance. If neighbouring frames are close to each other perceptually, then there the content does not change rapidly across the image, which means that it stitches well. We report this metric for different values of K on

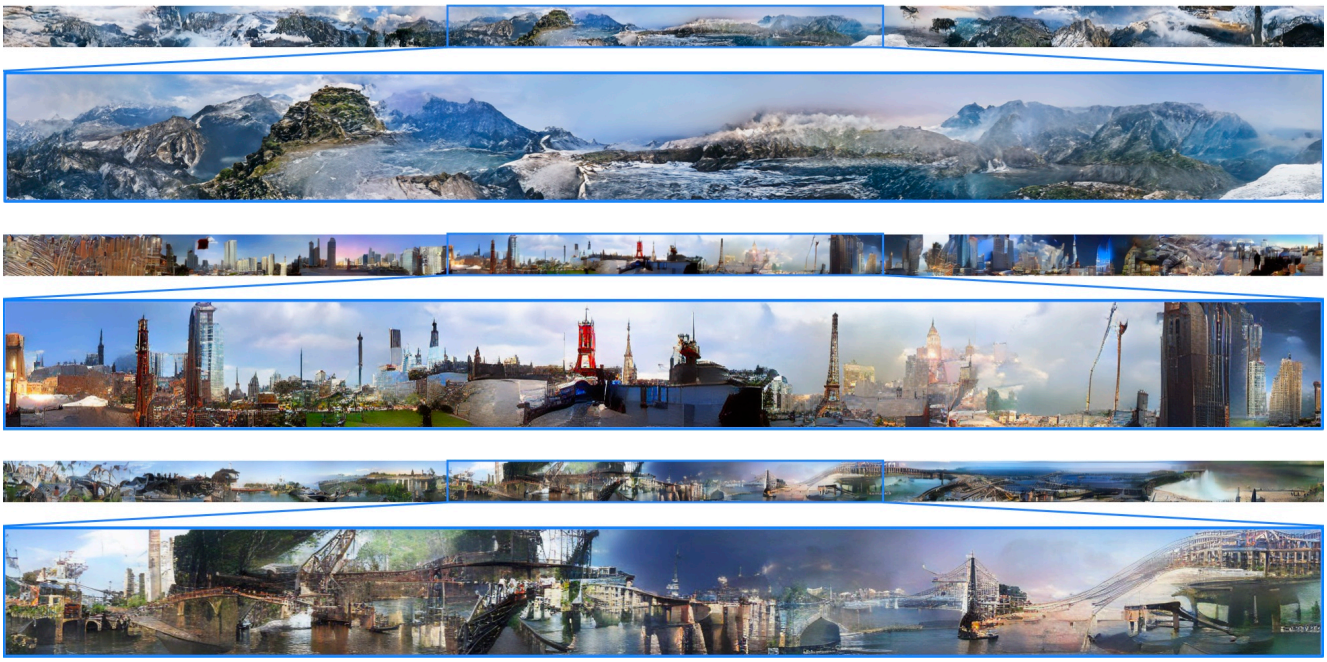


Figure 21: Random samples from Taming Transformer [10] on LHQ, LSUN Tower and LSUN Bridge

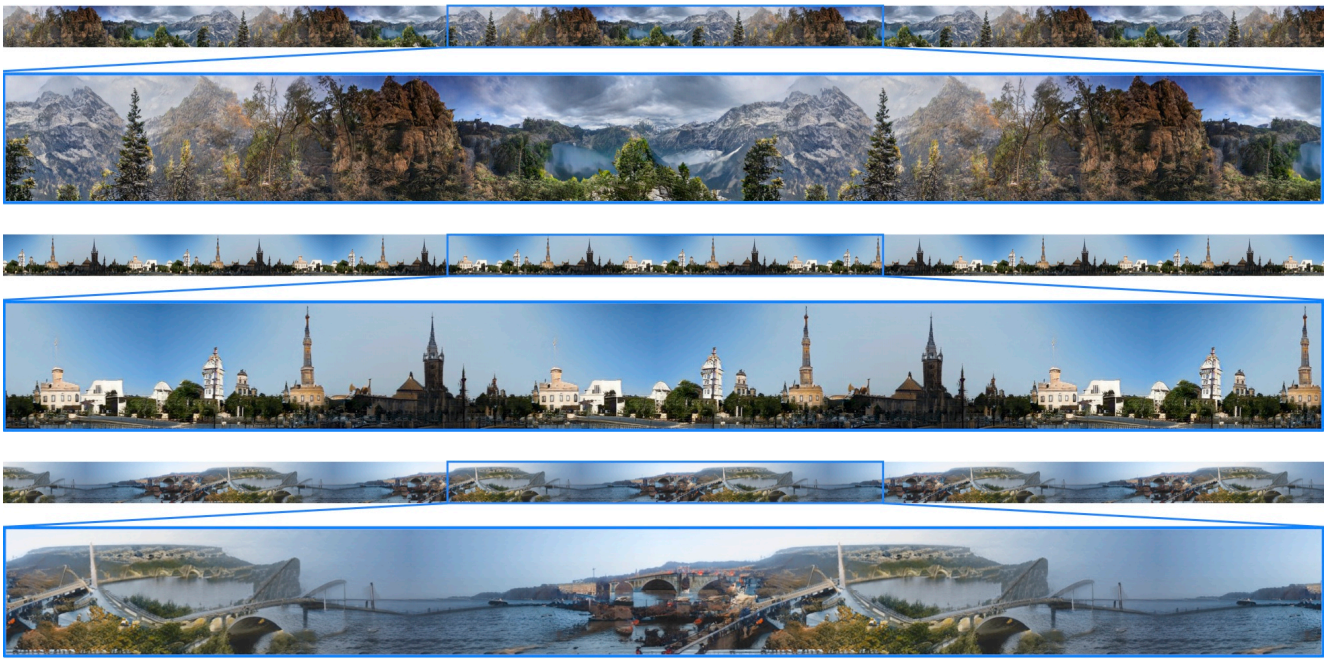


Figure 22: Random samples from LocoGAN+SG2+Fourier on LHQ, LSUN Tower and LSUN Bridge

Fig 27.

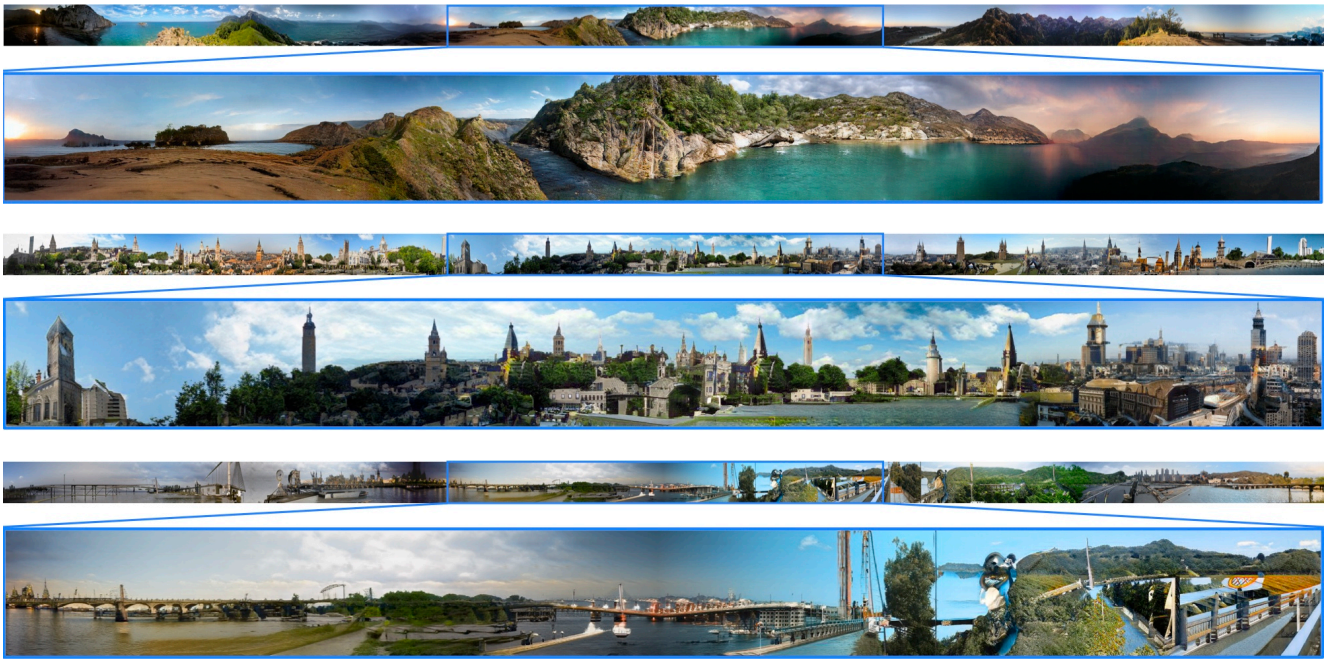


Figure 23: Random samples from ALIS on LHQ, LSUN Tower and LSUN Bridge

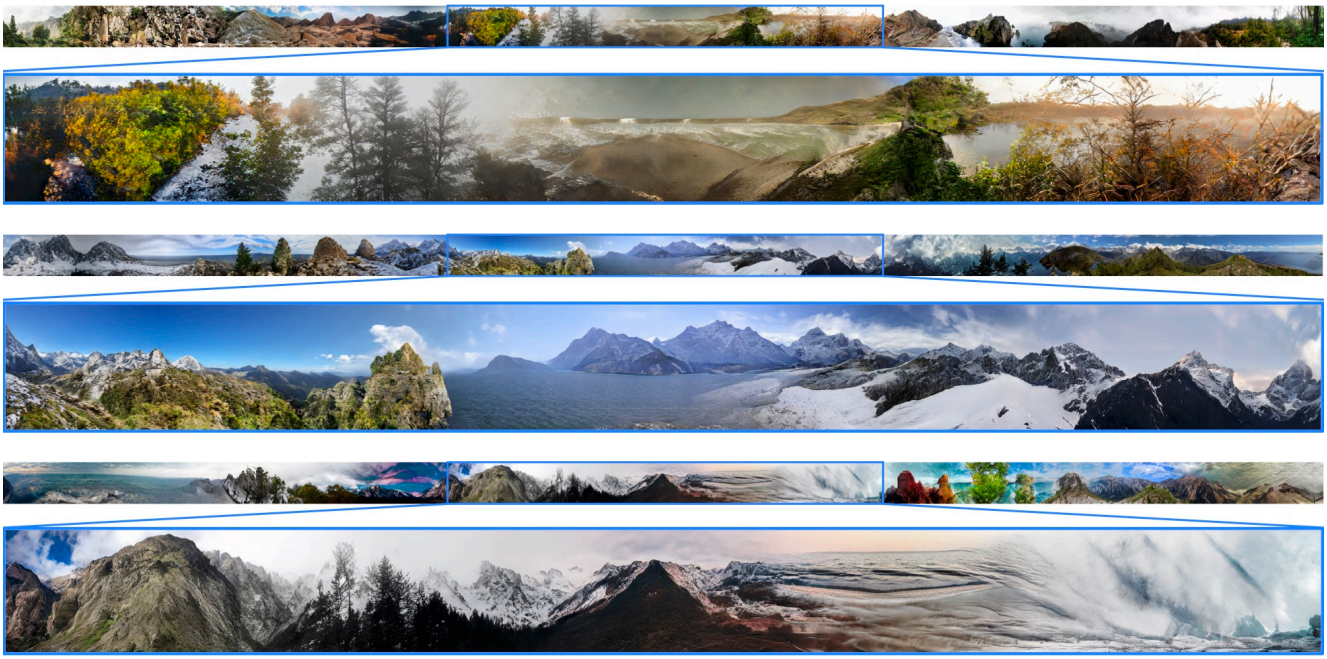


Figure 24: Random samples from ALIS on LHQ. Compare to clustered sampling in Figure 25

Figure 25: Random samples using "clustered sampling" on LHQ. Compare to random samples in Figure 24. To generate these images, we trained a Gaussian Mixture Model on 20k latent vectors with 8 components. Then, to generate a single long image, we sampled randomly from only a single mode. In this figure, samples from 4 different modes are presented. This improves sample quality since connecting too different anchors (like close-by water and far-away mountains) produces poor performance (see Figure 9). But this also decreases the diversity of samples.

Figure 26: Ablating coordinate embeddings for ALIS on LHQ, LSUN Tower and LSUN Bridge. As being discussed in Section 5, this leads to blurry generations.

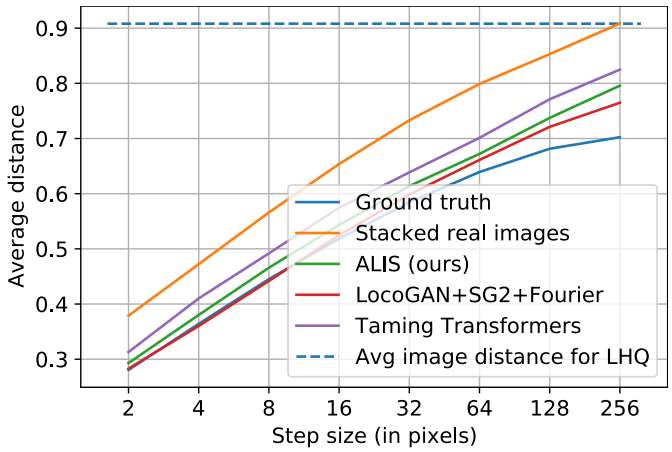


Figure 27: SPPL results on LHQ 256^2. For "ground truth", we computed SPPL using (sufficiently) horizontal images. For "stacked real images", we simply concatenated the entire LHQ into a single image.