

## Appendix

### A. Computational Complexity

The complexity of Byte Pair Encoding is determined by how many pairs of tokens have to be iterated over for counting and replacement. When applying BPE on an image with width and height  $W, H$ , it can either be flattened into a sequence of length  $W \cdot H$ , or processed as-is by counting row-wise. This leads to  $W \cdot H - 1$  pairs to count (flattened) or  $(W - 1) \cdot H$  (row-wise) possible pairs to count, *i.e.*  $\approx W \cdot H$  many pairs. When applying our approach, we count  $(W - 1) \cdot H$  horizontal pairs and  $W \cdot (H - 1)$  vertical ones, *i.e.*  $\approx 2(W \cdot H)$  pairs to count in total. The same argument holds for the replacement step, hence our overall increase in complexity is just a factor of 2 compared to BPE. The total complexity thus becomes  $2 \cdot |V| \cdot |W - 1| \cdot |H - 1|$ , with  $|V|$  as the vocabulary size.

Our (unoptimised) implementation in C++ allows to *e.g.* condense 50,000 images of  $16 \times 16$  tokens by adding 128 new tokens within less than 4 minutes on a single core with 3.7GHz. This describes datasets like CIFAR[34], SVHN[42], or MNIST[16]. For large datasets like ImageNet [15] compressed by a VQGAN from  $256 - by - 256$  pixels to  $32 - by - 32$  tokens, a single consumer-grade CPU core can hence increase the vocabulary size by about 25 tokens per hour. This can be dramatically be reduced by multi CPUs, or only counting on a subset. In relation to training times, the computational overhead is hence neglectable, *i.e.* less than 5% for all our experiments. Further optimisation would increase this gap approximately by an order of magnitude (*e.g.* just using 10 instead of 1 core already yields that factor, as we can fully parallelise our approach).

### B. Training details

For all experiments, we use SFAdam [13], an improved version of Adam [41] for transformer optimisation. We always run all experiments ourselves to make sure conditions (like optimiser) are the same, with always using the same foundation if we use *e.g.* a VQ-VAE [66] as base. For training the transformers, we use a learning rate of 0.0002 instead of 0.001 when training on compressed sequences for a slight boost in performance (if not specified otherwise). However, our results still hold when using the same learning rate for the examples we tried. We always pad all sequences in a batch to the length of the largest one using end-of-sequence tokens, posing an open angle for future improvements to not waste FLOPs when processing shortened sequences.

For FID computation on labelled datasets, we use 1000/50 samples pro class (CIFAR-10/MNIST), always comparing with the whole test/validation set; *e.g.* for CIFAR-10, generating 50,000 samples to compare to the 10,000

testset examples would be excessive and only makes a minor difference in the FID scores (the original FID paper[25] argues to use 50,000 in the light of large datasets ImageNet[15], which means to use only 500 per class; we hence use double of that for CIFAR/MNIST etc. classes). For SVHN and CelebA, we use 10000 unconditional samples. We use nucleus sampling with a threshold of 0.9 and no temperature adjustment (temperature 1.0) for all reported scores, but found the relative difference between runs to be largely invariant to the sampling strategy, *i.e.* while multinomial or topk sampling or adjusted temperature changed the results, the relative improvement between condensed sequence and original sequence stayed largely the same.

According to common practise [3, 32], we further always scale our learning rate with  $\frac{1}{\sqrt{\frac{N}{54000000}}}$ , where  $N$  is the number of parameters of our model, as our base model we used for fine-tuning was at 54m parameters.

We generally follow the settings of [21] for our encoder-only transformers using the standard pytorch implementation without dropout, split into a small(MNIST, SVHN, CIFAR) and a large sized group (CelebA, Quantised Celeb/CIFAR): For MNIST: 16 layers, 8 heads, 512 dimensions (54m parameters)

For SVHN: 16 layers, 8 heads, 512 dimensions (54m parameters); VQ-VAE with  $8^2$ ,  $K = 512$

For CIFAR: 16 layers, 8 heads, 512 dimensions (54m parameters); VQ-VAE with  $16^2$ ,  $K = 512$

For CelebA (VQ-VAE): 24 layers, 16 heads, 1024 dimensions (211m parameters); VQ-VAE with  $8^2$ ,  $K = 512$ ,  $128^2$  pixels

For CelebA (VQGAN): 24 layers, 16 heads, 1024 dimensions (211m parameters); VQGAN with  $16^2$ ,  $K = 2048$ ,  $128^2$  pixels

For Quantised Celeb/CIFAR: 24 layers, 16 heads, 1024 dimensions (211m params);  $32^2$  pixels

### C. Comparison to Pixel-Level BPE for Auto-Regressive Image Generation

For colour quantisation, we follow their idea: We use a resolution of  $32 \times 32$  pixels while discretising each colour channel to 10 values per R/G/B by dividing by 26, *i.e.* turning an RGB value of 255, 200, 146 into 9, 7, 5 = 975. We show the development of the FID values for quantised CelebA [40], together with examples, in Fig. 15. Note that we only use  $32 \times 32$  pixels here, as we only have Nvidia 2080 TI for our evaluation, and larger images do not fit the memory without compression through a VQ-VAE.

We further provide the FID development on CIFAR-10, with a VQ-VAE instead of colour quantisation, in Fig. 16.



Figure 14. Upper row: Generated image, lower row: Closest in Inception[65] feature vector of the train set.

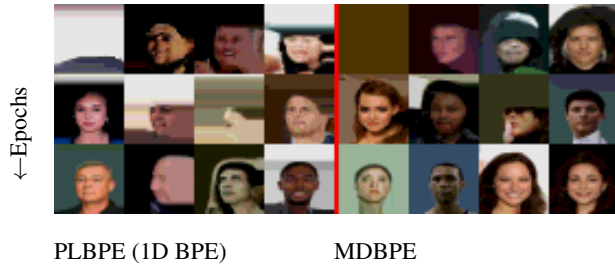
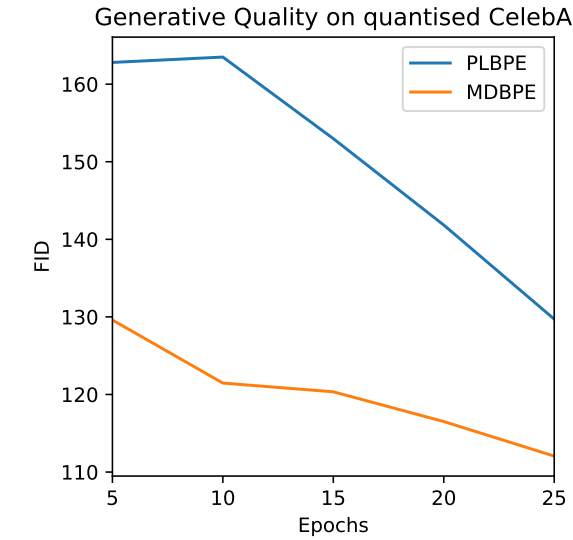


Figure 15. Generation over time on quantised CelebA at 32-by-32 pixels, comparing PLBPE [53] to our approach MDBPE, including improved embeddings. Examples show epochs 5, 10, 25.

## D. Experiment Specifications on Different Datasets

For all examples given in Tab. 2, we observe similar curves to Fig. 13, *i.e.* earlier saturation for shortened sequences and better overall performance. For SVHN [42] and MNIST [16], we trained for 30 epochs, while for the more complex CIFAR [34], we trained for 50 epochs. We trained 25 epochs for the relatively large CelebA [40].

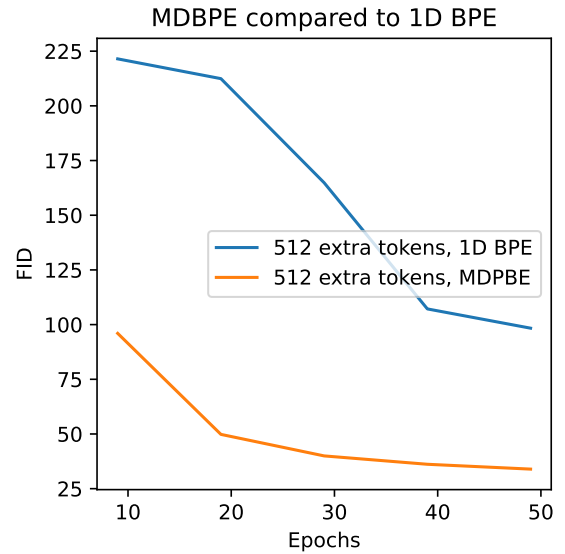


Figure 16. Development of FID values when applying MDBPE and the idea behind [53], namely, 1D BPE, on CIFAR-10 [34].

## E. Details on Generating Geometry

For geometry, we train the transformer for 500 epochs, as the dataset is much smaller, and use weight decay of 0.01. We use a transformer with 8m parameters each, with 256 dimensions, 8 layers, and 8 heads. For sampling, we use multinomial sampling with temperature 2.0 (the vanilla transformer would collapse to very few shapes without this adjustment).

For MMD and Coverage, we follow[1]: We split the data in train and test, with 20 percent of the data as test split, then generate as many examples as in the train split. We then compute coverage by finding the closest example for each generated object in the testset, then computing the percentage of the testset that is "covered" by an example as the nearest, measuring the fidelity of the distribution / the diversity. For MMD, we compute the distance of each testset object to the closest generated one, measuring

the faithfulness / the quality of the generated objects. As distance measure, we use chamfer distance on uniform random sampled point clouds of the extracted meshes from surface, with 2048 samples each. We always sample with a temperature of 1.75 for  $32^2$ , as is optimal for the 0-extra-token-case, for comparability.

We show generated examples of  $8 \times 8 \times 8$  and  $32 \times 32 \times 32$  in Fig. 17. We did not use the same classes everywhere for more variation and because *e.g.* the often very thin rifle park do not make much sense in  $8^3$ .

**SDFs** We applied a naive VQ-VAE[66] on a  $32^3$  distance field where we clamped the distance values to a fixed value for every cell further than two away from the surface (only voxels with the surface matter for the resulting quality), then apply our compression scheme with 512 extra tokens on the resulting  $8^3$  token grid where it achieved an average compression of 47%. We then generated these sequences as usual. Note that our results are not published here, we only add this as a further demonstration of versatility and are not experts on SDFs, *e.g.* the VQ-VAE has a lot of potential. Also, our SDF converter to obtain the SDFs failed on some of the chairs used, hence our scores would not be fair. We always use learning rate 0.00005 with temperature 1.5, and (due to the more rich information) use a bigger transformer with 16 layers, 1024 dimensions, and 16 heads.

## F. VQ-VAE Settings for Compression

We apply a VQ-VAE with latent size  $8^2$ ,  $K = 512$  for SVHN, a VQ-VAE with  $16^2$ ,  $K = 512$  for CIFAR-10 at  $32^2$  pixels and CelebA at  $128^2$  pixels, and a VQGAN for CelebA with latent size  $16^2$ ,  $K = 2048$ , and  $128^2$  pixels. Note that we always use the same VQ-VAE/VQGAN as baseline for all our experiments for comparability, *i.e.* all gains in performance stem only from our sequence shortening. For SDFs, we use a VQ-VAE with  $K = 512$  that reduces a  $32^3$  latent grid to  $8^3$ .

## G. Methodological Comparison to Other Approaches

We compare our method to other approaches on a methodological level: Approaches like [20, 73] that natively produce 1D sequences implicitly end up compressing/skipping redundant information like blue sky. However, they are not compatible with already existing tokens, and rely on intricate architecture designs. There is further no direct control/interpretability given, as our tokens can be directly visualised and analysed. Approaches like [28, 29, 54] perform a hierarchical approach, which leaves less fidelity to the token shapes: they do not allow for tokens that are not convex. Other approaches follow the idea of adaptivity based

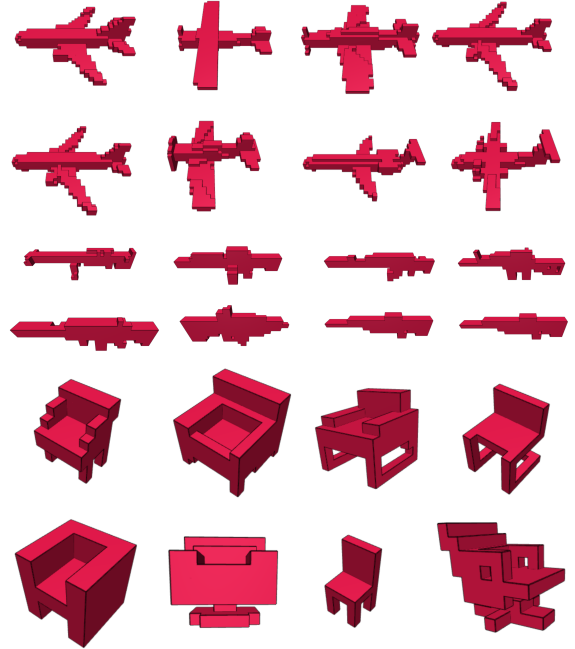


Figure 17. Generated voxel grids from ShapeNet [9] at a resolution of  $32^3$  (planes, rifles) and  $8^3$  (chairs) with our approach. Without compression, attention computation on  $32^3$  voxels for  $32^3$  would far exceed the memory of consumer GPUs.

on content [19, 59, 72], but all require a cut-off at which a loss in image quality is neglectable. This is contrary to our lossless compression. We only shift complexity from sequence size to vocabulary size.

The concurrent work of Zhang *et al.* [74] also applies an extended version of BPE on tokens, but does so with a focus on more specific tasks instead of the general task of generation and follows a more intricate procedure to obtain compacted tokens. Compared to the compression of similar tokens [6], our work is more general: We can even compress different tokens together that do frequently co-occur, not only merge areas of similar content.

Compared to all of these other approaches, our work makes use of a large vocabulary in a lossless way: Transformers have shown a strong preference for large vocabularies over larger sequences (*e.g.*  $|V| = 128.000$  for [18]).

## H. Diversity in Shortened Sequences

We follow the approach of Esser *et al.* [21] and find the nearest image in the sense of euclidean distance in the feature vectors of the images from the second last layer of the Inception network[65] from the train dataset for randomly picked generated images in Fig. 14.

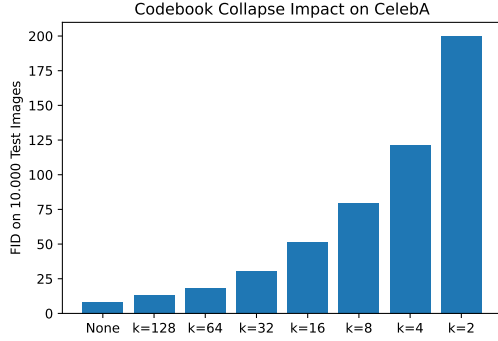


Figure 18. Impact of the number of tokens used, *i.e.* how the codebook collapse barely impacts quality.

## I. Step-by-Step Example of MDBPE vs 1D BPE

We show a step-by-step application of 1D BPE versus MDBPE in Fig. 19. In practise, we observe that for the most part, larger empty areas (dark background parts, blue sky, larger areas like the yellow beak of a toucan) are put into tokens. However, opposing to *e.g.* [6] where only such similar regions are compressed, we also observe cases of very different token combinations being compressed together (*e.g.* very bright superpixels next to very dark superpixels are frequently combined). It also varies greatly on the dataset: On MNIST, we can see tokens forming an entire “cut-out” for numbers (a coarse inverse of the number 3, left haft), as the black background is easier to combine than the numbers with possibly slight differences in brightness.

## J. Classification

We demonstrate that shorter sequences not only benefit generative tasks, but also classification. We hence do a small example of classification on CIFAR, comparing compressed versus uncompressed sequences in Fig. 20. Our saturates faster, as (in this case) shorter sequences apparently also work better for encoder-only transformers like ViTs [17].

## K. Reducing Tokens in VQGANs

We evaluate our proposed VQGAN collapse in two different ways: We show qualitatively how the choice of  $K$  barely impacts the result in Fig. 21, exemplary on ImageNet as a more general dataset [15]. We then measure the quantitative impact of the number of tokens used for images of human faces of CelebA [40] in Fig. 18: Using only 128 instead of the full 2048 codebook entries can produce almost the same fidelity. We mostly observe small changes in the high frequent details of an image for (relatively) high choices of  $K$ , *e.g.* small earrings appearing and disappearing, that barely affect quality. We also argue that a more powerful

VQGAN (ours only uses 28m parameters, which is orders of magnitude smaller than the state of the art) can probably make up for this even better, and that further fine-tuning on the results would also be beneficial. We leave this for future work, as our focus is on compression, not on tuning VQGANs (and the Nvidia 2080 TI our lab is equipped with barely fit a batch of high resolution images already when only using 28m parameters).

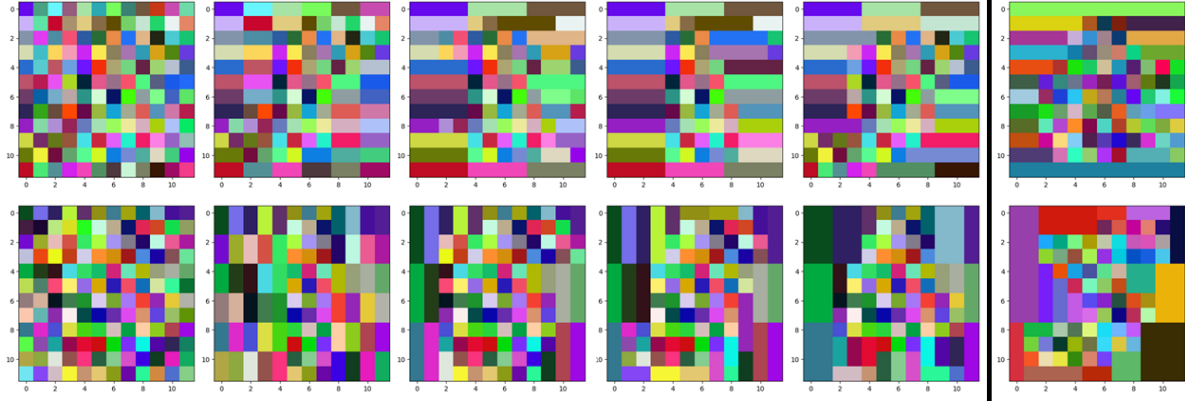


Figure 19. Visualisation of the resulting byte-pair encoding after 1/2/3/4/5/32 steps on MNIST at 12-by-12 pixels for better visibility.

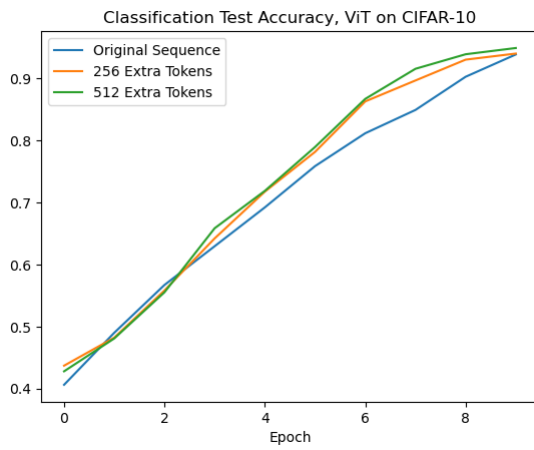


Figure 20. Result of our work on classification tasks on CIFAR-10. Y-Axis showing accuracy of classification.



Input

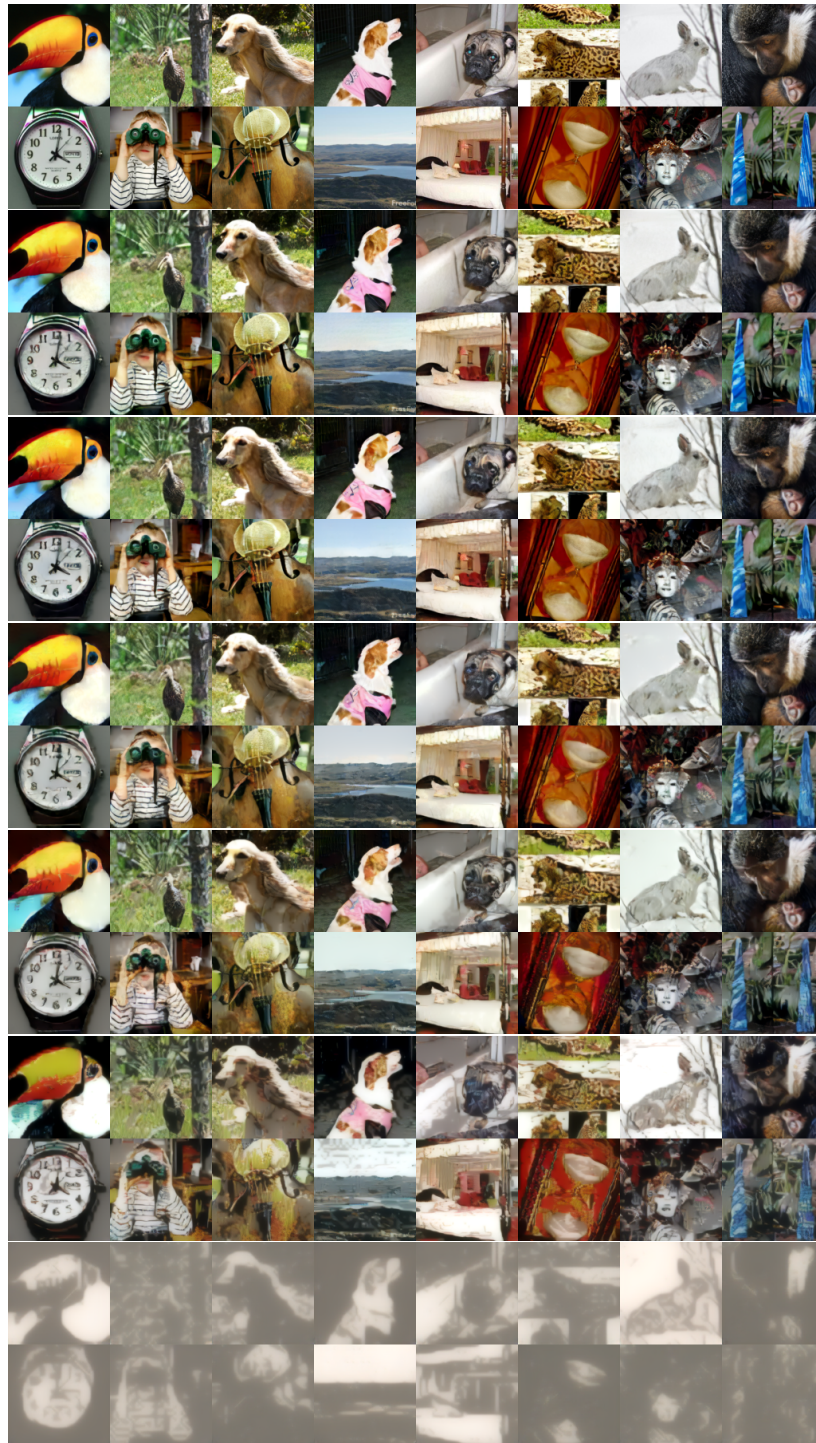
 $K = 128$  $K = 64$  $K = 32$  $K = 16$  $K = 8$  $K = 2$ 

Figure 21. Different numbers of codebook entries  $K$  and their impact on the resulting image quality produced by a VQGAN [21] with initially 2048 codebook entries. Note how dropping from 2048 to 128 tokens in posterior barely makes a visible difference.