Online Clustered Codebook

A. Experiment Details

For data compression, we first demonstrate our method on small datasets with the officially released VQ-VAE [37] implementation [45], and we then verify the generality of our quantiser on large datasets using the officially released VQ-GAN [11] architecture [6]. For image generation application, we apply our CVQ-VAE's quantiser on LSUN dataset using the officially released LDM [32] code[7].

For the small datasets (MNIST, CIFAR10, and Fashion MNIST), we use the submitted code to train the model. The training hyperparameters match the original VQ-VAE settings, and we train all models with batch size 1,024 across 4× NVIDIA GeForce GTX TITAN X (12GB per GPU) with 500 epochs (2-3 hours).

For the high resolution datasets (FFHQ and ImageNet), we just replace the original quantiser in VQGAN with our CVQ-VAE quantiser. The training hyperparameters also follow the original settings, and we train all models with batch size 64 across 4× NVIDIA RTX A6000 (48GB per GPU) with 4 days on FFHQ and 8 days on ImageNet until converge.

For the generation (LSUN bedrooms and Churches), we use the lSUN-beds256 config file for default setting with two modifications: 1) we also replace the VQGAN's quantiser with our CVQ-VAE quantiser; 2) we reduce the images' resolution for faster training with 8× downsampling. For stage **a)** codebook learning, two models are trained with batch size 32 across 2× NVIDIA RTX A4000 (48GB per GPU) with 5 days. Then, for stage **b)** latent diffusion model with $32 \times 32 \times 4$ resolution, we train the models with batch size 128 across 2× NVIDIA RTX A4000 (48GB per GPU) with 7 days. During the inference, we follow the default settings to sample the images with 200 steps.

B. Quantitative Results

Method	**Dataset**	ℓ_1 loss ↓	SSIM ↑	PSNR ↑	LPIPS ↓	rFID ↓
VQ-VAE [37]	MNIST	0.0207	0.9777	26.48	0.0282	3.43
HVQ-VAE [39]		0.0202	0.9790	26.90	0.0270	3.17
SQ-VAE [36]		0.0197	0.9819	27.49	0.0256	3.05
CVQ-VAE		**0.0180**	**0.9833**	**27.87**	**0.0222**	**1.80**
VQ-VAE [37]	CIFAR10	0.0527	0.8595	23.32	0.2504	39.67
HVQ-VAE [39]		0.0533	0.8553	23.22	0.2553	41.08
SQ-VAE [36]		0.0482	0.8779	24.07	0.2333	37.92
CVQ-VAE		**0.0448**	**0.8978**	**24.72**	**0.1883**	**24.73**

Table B.1: **Reconstruction results** on the validation sets of MNIST (10,000 images) and CIFAR10 (10,000 images).

Table B.1 provides a comparison of our results to state-of-the-art quantisers under the same training settings, except for the different quantisers, on the small datasets. This is an extension of Tab. 1 in the main paper. All images are normalised to the range [0,1] for quantitative evaluation. See the code for more details. While the proposed CVQ-VAE achieve relative small improvements on traditional pixel-level ℓ_1 loss, peak signal-to-noise ration (PSNR), and patch-level structure similarity index (SSIM), it significantly improves the feature-level LPIPS and dataset-level rFID, suggesting that our CVQ-VAE is more capable of reconstructing the content closer to the dataset distribution.

We further compare our CVQ-VAE to the state-of-the-art methods in data compression in Tab. B.2. This is an extension of Tab. 2 in the main paper. Here, we add the pixel-level PSNR, patch-level SSIM and feature-level LPIPS. For FFHQ dataset, our CVQ-VAE model outperforms baseline variants of previous state-of-the-art models. As for ImageNet dataset, while our 4× channels setting does not achieve the better rFID than the latest MoVQ model, the other instantiations (PSNR, SSIM and LPIPS) significantly outperform existing state-of-the-art models.

Tables B.3 and B.4 are the extension of Tabs. 3 and 4c in the main paper, respectively. Even reported with the different metrics, The conclusions are still the same. For instance, the offline version is significantly affected by different anchor

[4] https://github.com/deepmind/sonnet/blob/v2/sonnet/src/nets/vqvae.py

[5] https://github.com/deepmind/sonnet/blob/v1/sonnet/examples/vqvae_example.ipynb

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

[7] https://github.com/CompVis/latent-diffusion

Method	**Dataset**	$\mathcal{S}\downarrow$	$\mathcal{K}\downarrow$	Usage $\uparrow$	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$	rFID $\downarrow$
VQGAN [11]	FFHQ	16^2	1024	42%	22.24	0.6641	0.1175	4.42
ViT-VQGAN [44]		32^2	8192	–	–	–	–	3.13
RQ-VAE [22]		$16^2\times4$	2048	–	22.99	0.6700	0.1302	3.88
MoVQ [47]		$16^2\times4$	1024	56%	26.72	0.8212	0.0585	2.26
SeQ-GAN [13]		16^2	1024	100%	–	–	–	3.12
CVQ-VAE (ours)		16^2	1024	100%	26.82	0.8313	0.0608	2.80
CVQ-VAE (ours)		$16^2\times4$	1024	100%	**26.87**	**0.8398**	**0.0533**	**2.03**
VQGAN [11]	ImageNet	16^2	1024	44%	19.07	0.5183	0.2011	7.94
ViT-VQGAN [44]		32^2	8192	96%	–	–	–	1.28
RQ-VAE [22]		$8^2\times16$	16384	-	–	–	–	1.83
MoVQ [47]		$16^2\times4$	1024	63%	22.42	0.6731	0.1132	**1.12**
SeQ-GAN [13]		16^2	1024	100%	–	–	–	1.99
CVQ-VAE (ours)		16^2	1024	100%	21.95	0.6612	0.1340	1.57
CVQ-VAE (ours)		$16^2\times4$	1024	100%	**23.37**	**0.7115**	**0.1099**	1.20

Table B.2: **Reconstruction results** on validation sets of ImageNet (50,000 images) and FFHQ (10,000 images). $\mathcal{S}$ denotes the latent size of encoded features, and $\mathcal{K}$ is the number of codevectors in the codebook.

	Method	**MNIST** (28×28)			**CFAIR10** (32×32)			**Fashion MNIST** (28×28)		
		$\ell_1\downarrow$	PSNR $\uparrow$	rFID $\downarrow$	$\ell_1\downarrow$	PSNR $\uparrow$	rFID $\downarrow$	$\ell_1\downarrow$	PSNR $\uparrow$	rFID $\downarrow$
$\mathbb{A}$	Baseline VQ-VAE [37]$_{\text{NeurIPS'2017}}$	0.0207	26.48	3.43	0.0527	23.32	39.67	0.0377	23.93	12.73
$\mathbb{B}$	+ Cosine distance	0.0200	26.77	3.06	0.0509	23.66	35.14	0.0378	24.01	11.40
$\mathbb{C}$	+ Anchor initialization (offline)	0.0192	27.24	2.78	0.0481	24.16	31.10	0.0373	24.04	11.92
$\mathbb{D}$	+ Anchor initialization (online)	0.0186	27.58	2.23	**0.0445**	**24.79**	26.62	0.0349	**24.69**	9.27
$\mathbb{E}$	+ Contrastive loss	**0.0180**	**27.87**	**1.80**	0.0448	24.72	**24.73**	**0.0344**	24.66	**8.85**

Table B.3: **Results on various settings.** We add pixel-level ℓ_1 and PSNR metrics.

Method	**Dataset**	Offline					Online				
		$\ell_1$loss $\downarrow$	SSIM $\uparrow$	PSNR $\uparrow$	LPIPS $\downarrow$	rFID $\downarrow$	$\ell_1$loss $\downarrow$	SSIM $\uparrow$	PSNR $\uparrow$	LPIPS $\downarrow$	rFID $\downarrow$
random	MNIST	0.0195	0.9802	27.11	0.0262	3.20	**0.0185**	**0.9823**	**27.58**	**0.0236**	2.27
unique		0.0191	0.9811	27.25	0.0255	2.84	0.0186	0.9820	27.51	0.0237	2.24
probability		0.0192	0.9810	27.24	0.0253	2.78	0.0186	**0.9823**	**27.58**	**0.0236**	**2.23**
closest		**0.0186**	**0.9823**	**27.59**	**0.0242**	**2.51**	0.0187	0.9819	27.49	0.0244	2.59
random	CIFAR10	0.0494	0.8755	23.91	0.2256	34.49	0.0440	**0.9010**	24.88	0.1881	26.04
unique		0.0507	0.8705	23.15	0.2346	36.99	**0.0439**	0.9007	**24.91**	**0.1877**	26.03
probability		**0.0481**	**0.8829**	**24.16**	**0.2131**	**31.10**	0.0445	0.8991	24.79	0.1898	26.62
closest		0.0487	0.8804	24.06	0.2156	32.31	0.0444	0.8994	24.83	0.1900	**25.99**

Table B.4: **Anchor sampling methods.** The choice of anchor sampling method has a significant impact on offline (one-time) feature initialization, while the online clustered method is robust for various samplings.

sampling methods, but the online version is not sensitive to various anchor sampling methods. The online version holds very close performance with these anchor sampling methods.