A APPENDIX

A.1 EXPERIMENT HYPERPARAMETERS

This section contains additional experimental details and hyperparameters. All models were implemented in PyTorch 1.12 distributed by Paszke et al. (2019). Several assets were forked from Esser et al. (2021); Vahdat & Kautz (2020) and Sajjadi et al. (2018) to assist with benchmarking.

Group	Hyperparameter	FashionMNIST	CIFAR10	CelebA64	Imagenet64
General	Batch Size	32	32	32	32
	Epochs	350	650	250	650
Optimizer	Name	Adam	Adam	Adam	Adam
	Learning Rate	$5e{-}4$	$5e{-}4$	$5e{-}4$	$5e{-}4$
	Betas	$0.5, 0.9$	$0.5, 0.9$	$0.5, 0.9$	$0.5, 0.9$
Codebook (2)	Units ($D + D'$)	$16 + 16$	$96 + 96$	$256 + 256$	$256 + 256$
	Beta	0.25	0.25	0.25	0.25

Table 2: Hyper-parameters for every model across each dataset.

Down-sampling factor (F)		7/8	14	16	28	32	64
Encoder 1 ($\mathcal{E}$)	Kernels	4,3,3,3	4,3,3,3,3	4,4,4,3,3	4,4,4,4,3	4,3,3,3,3,3	4,4,4,4,4,4
	Channels	128	128	128	128	128	128
	Residual Blocks	2	2	2	2	2	2
	Residual Kernels	3,1	3,1	3,1	3,1	3,1	3,1
	Residual Channels	64	64	64	64	64	64
Encoder 2 ($\mathcal{E}'$)	Kernels	4,3,3	4,3	4,3,3	4,3	4,3	3,1
	Channels	128	128	128	128	128	128
	Residual Blocks	2	2	2	2	2	2
	Residual Kernels	3,1	3,1	3,1	3,1	3,1	3,1
	Residual Channels	64	64	64	64	64	64
Decoder 1 ($\mathcal{D}$)	Kernels	3,3,4,4	2,2,3,4,4	3,4,4,4,4	1,2,3,4,4	2,2,4,4,4,4	2,4,4,4,4,4
	Channels	128	128	128	128	128	128
	Residual Blocks	2	2	2	2	2	2
	Residual Kernels	3,1	3,1	3,1	3,1	3,1	3,1
	Residual Channels	64	64	64	64	64	64
Decoder 2 (f)	Kernels	2,2,4	2,1	2,1	2,1	2,1	2,1
	Channels	128	128	128	128	128	128
	Residual Blocks	2	2	2	2	2	2
	Residual Kernels	3,1	3,1	3,1	3,1	3,1	3,1
	Residual Channels	64	64	64	64	64	64

Table 3: Encoder and decoder based on VQVAE2 (Razavi et al., 2019) with batch normalisation between layers. Encoders use convolutional layers with intermediate ReLU activation layers. Decoders use one convolutional layer followed by deconvolutional layers with LeakyReLU activations.

#Tokens per Instance		1 + 1	4 + 1	16 + 1
Primary Transformer	Layers	4	8	12
	Heads	4	8	8
	Embed Units	64	128	256
	Parameters (M)	0.2	1.6	9.5
Secondary Transformer	Layers	4	4	4
	Heads	4	4	4
	Embed Units	64	128	256
	Parameters (M)	0.2	0.8	3.2

Table 4: Hyper-parameters for probabilistic models used to learn the categorical posterior.

A.2 Additional Metrics

		Tokens per Instance	1 + 1		4 + 1		16 + 1	
Metric	Dataset	Codebook Entries	16	256	16	256	16	256
FID*↓	CIFAR10	VQGAN	230.79	158.76	69.25	46.86	**46.24**	**41.54**
		MVQGAN	**55.16**	**51.81**	**50.19**	**41.72**	47.42	46.23
	CelebA64	VQGAN	114.82	89.65	35.05	29.80	29.83	29.02
		MVQGAN	**34.07**	**31.53**	**31.14**	**29.05**	**26.87**	**25.69**
	ImageNet64	VQGAN	286.01	181.02	154.25	81.27	63.57	62.52
		MVQGAN	**85.04**	**74.26**	**65.52**	**61.84**	**61.79**	**60.50**
Primary NLL	CIFAR10	VQGAN	1.307	2.441	1.990	2.289	2.011	2.251
		MVQGAN	2.043	4.073	1.906	4.284	1.879	4.458
	CelebA64	VQGAN	1.697	1.934	1.885	3.484	1.906	2.575
		MVQGAN	2.065	4.004	1.990	4.263	1.761	3.843
	ImageNet64	VQGAN	1.841	2.121	1.984	2.537	1.738	1.650
		MVQGAN	1.931	3.940	1.876	4.289	1.798	3.711
Secondary NLL	CIFAR10	VQGAN	1.074	2.474	1.825	3.536	1.741	2.730
		MVQGAN	2.015	2.415	1.765	2.700	1.834	3.012
	CelebA64	VQGAN	1.960	2.895	2.032	3.789	1.838	3.987
		MVQGAN	1.882	2.940	1.650	1.527	1.433	1.743
	ImageNet64	VQGAN	1.963	3.590	1.789	3.990	1.810	3.927
		MVQGAN	1.590	2.311	1.503	2.483	1.523	1.858
$F_{\beta=1/8}$ ↑	CIFAR10	VQGAN	0.152	0.643	0.741	0.835	0.855	**0.904**
		MVQGAN	**0.746**	**0.752**	**0.883**	**0.890**	**0.866**	0.896
	CelebA64	VQGAN	0.280	0.452	0.778	0.791	0.857	0.869
		MVQGAN	**0.757**	**0.757**	**0.792**	**0.827**	**0.882**	**0.905**
	ImageNet64	VQGAN	0.073	0.327	0.311	0.582	0.633	0.652
		MVQGAN	**0.590**	**0.660**	**0.652**	**0.658**	**0.676**	**0.718**

Table 5: Sample FID (FID*), negative log-likelihood (NLL) and F-score for validation split.

	Down Sampling Factor ($\mathcal{F}$)		Time to Sample (s)	
#Tokens per Image	256x256	64x64	GPU	CPU
1024 + 1	8	2	511.047	1365.668
256 + 1	16	4	14.844	268.368
64 + 1	32	8	0.718	11.107
16 + 1	64	16	0.098	0.637
4 + 1	128	32	0.031	0.070
1 + 1	256	64	0.013	0.014

Table 6: Comparisons of token sampling times across consumer grade hardware: Nvidia RTX2080 (GPU) and Intel i9-9900 (CPU)

A.3 RECONSTRUCTION QUALITY

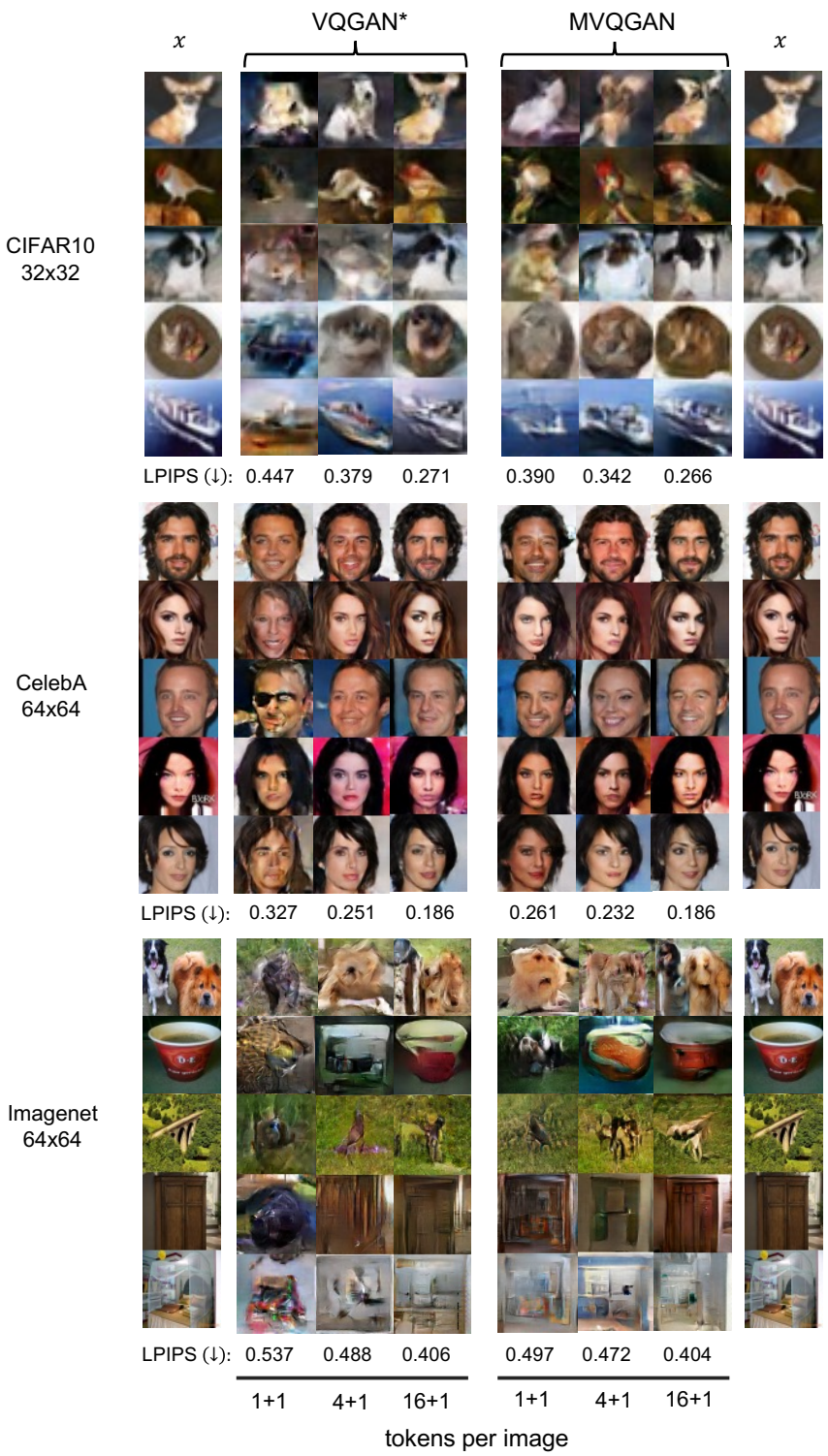

Figure 7: Reconstruction of instances (validation) using 256 codebook entries.

A.4 DIVERSITY OF MASKED CONFIGURATIONS OF THE SAME CODE VECTOR

Figure 8: MVQGAN with 256 codebook entries. Each 2×4 or 2×5 block represents masked configurations of the same code vector. This demonstrates that the diversity decreases as the number of token per image increases.

A.5 REPRESENTATIONAL TRANSFER

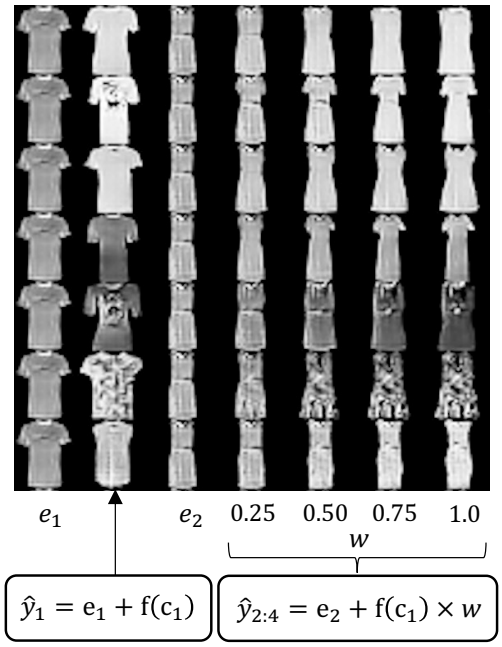

Figure 9: MVQGAN (K=4, S=1+1) demonstrating representational transfer on FashionMNIST. This demonstrates the transfer of patterns, tones, and shapes from centroid e_1 to e_2.

A.6 COMPARISON TO VQVAE2

In this section, we discuss the similarities and differences between the MVQ and VQVAE2 architecture. The major difference is that MVQ finds a mask for the secondary latent representation for each instance (Sec 3.2). The mask is the best of a randomised search of possible masks as determined by reconstruction error in MH-Dropout (Sec 3.3).

There are similarities between MVQ's and VQVAE2's primary-secondary pipelines, however there two main differences:

1. Secondary Encoder: As per Algorithm 1 (step 3) in Razavi et al. (2019), the secondary encoder takes as input both the instance $\mathbf{x}$ and output of the primary encoder. As per Figure 3, the MVQ secondary encoder only takes the output of the primary encoder.
2. Decoder: As per Algorithm 1 (step 5) in Razavi et al. (2019), the VQVAE2 decoder accepts a concatenated primary and secondary code vector. As per Figure 3 and Equation 5, the MVQ decoder accepts the summation of the primary code vector and masked secondary code vector.