A. Buffer Memory Management

We consider two settings to manage the samples inside the memory. In the first setting, we do not perform any codebook freezing. Therefore, as the model trains, the amount of compressed representations AQM can store increases smoothly. In this setting, the current amount of samples stored by AQM is a good approximation of the model's capacity. Therefore, we simply use this estimate instead of the total buffer size in the regular reservoir sampling scheme. The algorithm is presented in Alg 4.

Algorithm 4: AddToMemory

```
Input: Memory $\mathcal{M}$ with capacity $C$ (bytes), sample $x$
$N_{reg} = \frac{C}{BYTES(x)}$
capacity = max( $N_{reg}$, NUM SAMPLES ($\mathcal{M}$) )
 %Probability of adding x
add $\sim \mathcal{B}(\frac{\text{capacity}}{\text{SAMPLE AMT SEEN SO FAR}})$ %Bernoulli
 if add then
    $hid_x$, $block_{id}$ = ADAPTIVE COMPRESS($x$, $AE$, $d_{th}$)
    while BITS($hid_x$) - FREE SPACE($\mathcal{M}$) $> 0$ do
        DELETE RANDOM($\mathcal{M}$)
    end
end
```

This is not the case when we perform codebook freezing. In the latter setting, consider the moment when the first codebook is fixed; suddenly, the amount of samples the model can store has increased by a factor equal to the compression rate of the first block. Therefore, at this moment, the amount of samples currently stored by AQM is not a good approximation for the model's capacity.

Moreover, when performing codebook freezing, since the capacity suddenly spikes, we must decide between a) having an imbalance in the buffer, where certain temporal regions of the streams are not equally represented, or not utilising all available memory and storing less incoming samples so they are in similar quantities as previous samples. We opt for the former approach, and propose a procedure that allows the buffer to rebalance itself as new training data becomes available. We illustrate the procedure with an example.

Consider an AQM where the distribution of samples in the buffer is the one plotted in Fig 10. Specifically, we show the number of samples stored for each minibatch processed by the model. In this example, very few (<20) samples are stored from the earliest part of the stream, while a much larger number comes from the more recent part of the stream. Assuming that the model is over its memory capacity, we need to remove samples until the memory requirement is met. Ideally, we would like to remove more samples from parts of the stream where samples are abundant. In order to do so, we use Kernel Density Estimation on the histogram in Fig 10. Doing so gives us the line labelled `iter0` in Fig 10. We then sample points according to the distribution given by `iter0`, remove them, and fit a new KDE with the remaining points (labelled `iter1`). In this example we repeat this procedure 10 times, until `iter9`. As we can see, the distribution of stored samples becomes closer to the uniform distribution, i.e. the setting where all parts of the streeam are equally represented.

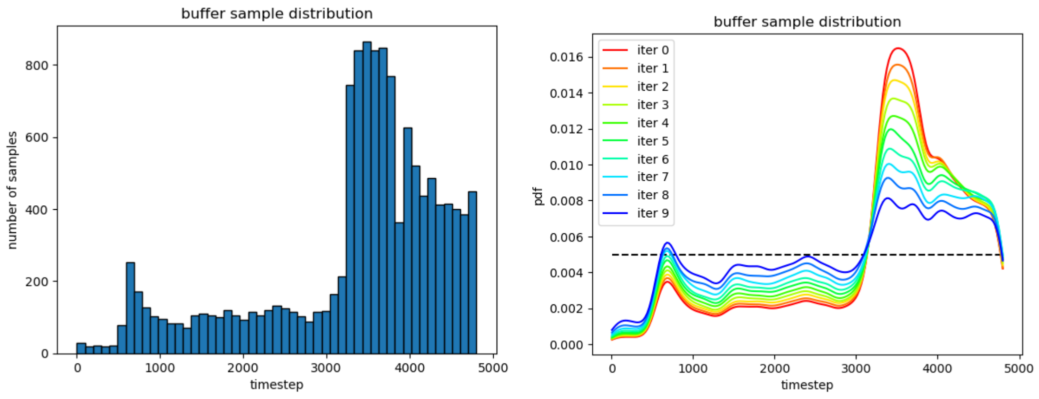

Figure 7. (left) histogram of samples in AQM where no buffer balancing was performed. (right) iterative buffer balancing procedure

Therefore, in the setting where codebook freezing is performed, we first add all incoming points to the buffer. Then, points are removed according to the procedure described above. This allows for maximal memory usage while ensuring that the buffer self balances over time. Note that we need to store the timestamp alongside each sample, which has a negligible cost when dealing with high-dimensional inputs.

B. Further Details of Experiments

We include here further details regarding the models used in our experiments. For all reported results, **almost all hyperparameters are kept the same.** we set D the size of the embedding table equal to 100, we use a decay value of 0.6 for the Embedding EMA update, and the same architectural blocks. Across problems, we mainly vary the reconstruction threshold, as well how the blocks are stacked and their compression rates (by e.g. changing the number of codebooks per block).

B.1. Cifar

For CIFAR-10, we use a 1 block AQM, latent size (16 x 16 x 100) is quantized with (16 x 16 x 1) indices where the last index represents the number of codebooks. The codebook here contains 128 embeddings, giving a compression factor of 13.7×. Due to the simplistic nature of the task and low resolution of the images, AQM already yields good compression before the end of the first task, hence adaptive compression and codebook freezing are not required.
For the (Riemer et al., 2018) baseline, we ran a hyperparameter search to vary the compression size. Specifically, we ran a grid search for the number of categories per latent variable, as well as for the number of latent variables. We found the gumbel softmax much less stable during training and harder to cross-validate than vector quantization.

Below we show an example of the image quality of our approach compared to (Riemer et al., 2018). We ran both AQM and (Riemer et al., 2018) on the split CIFAR-10 task, then extracted images which happened to be in the buffer of both methods.

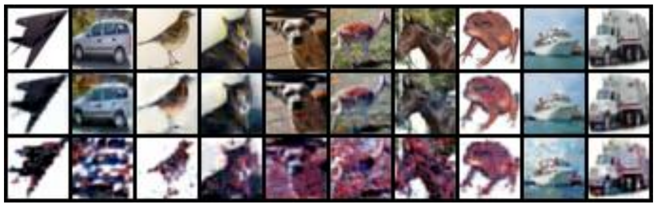

Figure 8. Bottom row: random buffer reconstructions using (Riemer et al., 2018). Middle row: random buffer reconstructions using SQM. Top row: corresponding original image. Columns are ordered w.r.t their entry time in the buffer, from oldest to newest. All samples above were obtained from the disjoint CIFAR-10 task, and are 12× smaller than their original image.

B.2. Imagenet

For the Offline 128 x 128 Imagenet experiment, we use the following three blocks to build AQM, with the following latent sizes :

1. (64 x 64 x 100) quantized using (64 x 64 x 1) indices, with a codebook of 16 vectors, giving a 24× compression.
2. (32 x 32 x 100) quantized using (32 x 32 x 1) indices, with a codebook of 256 vectors, giving a 48× compression.
3. (32 x 32 x 100) quantized using (32 x 32 x 1) indices, with a codebook of 32 vectors, giving a 76.8× compression.

For the 2 block AQM, we searched over using blocks (1-2) (2-3) and (1-3). For 1 block AQM we simply tried all three blocks independently.

When stacking two blocks with the same latent sizes (e.g. block 2 and 3) the encoder and decoder functions for the second block are the identity. In other words, the second block simply learns another embedding matrix.

C. Drift Ablation

Here we provide additional results for the drift ablation studied in 4. We repeat the same experiment for different values of the reconstruction threshold parameter, which controls when the codebook freezing occurs. The table below shows that the same conclusion holds across multiple values for this parameter: codebook freezing yields little to no drift, while only negligibly hindering the model's ability to adapt to a distribution shift.

It is worth noting that in cases of severe drift (e.g. with `recon th` $= 7.5$) the model diverges because it is rehearsing on samples of poor quality. In this setting, codebook freezing performs better on both the streaming MSE and the drift MSE.

`recon th`	Freezing	Streaming MSE	Drift MSE	Streaming + Drift MSE
1	No	0.59 ± 0.03	1.01 ± 0.14	1.60 ± 0.17
1	Yes	0.60 ± 0.03	0.49 ± 0.02	$\mathbf{1.09 \pm 0.05}$
2.5	No	0.63 ± 0.06	13.24 ± 5.36	13.87 ± 5.42
2.5	Yes	0.81 ± 0.02	1.07 ± 0.05	$\mathbf{1.88 \pm 1.12}$
5	No	0.65 ± 0.04	55.31 ± 36.82	55.96 ± 36.86
5	Yes	0.92 ± 0.03	1.69 ± 0.18	$\mathbf{2.61 \pm 0.21}$
7.5	`nan`	`nan`	`nan`	`nan`
7.5	Yes	0.98 ± 0.11	2.10 ± 0.28	$\mathbf{3.08 \pm 0.39}$

Table 4. Here we provide additional results for the drift ablation discussion in section shown in Fig 4. For clarity all results are multiplied by 100. (e.g. `recon th` of 2.5 corresponds to 0.025). Results averaged over 5 runs.

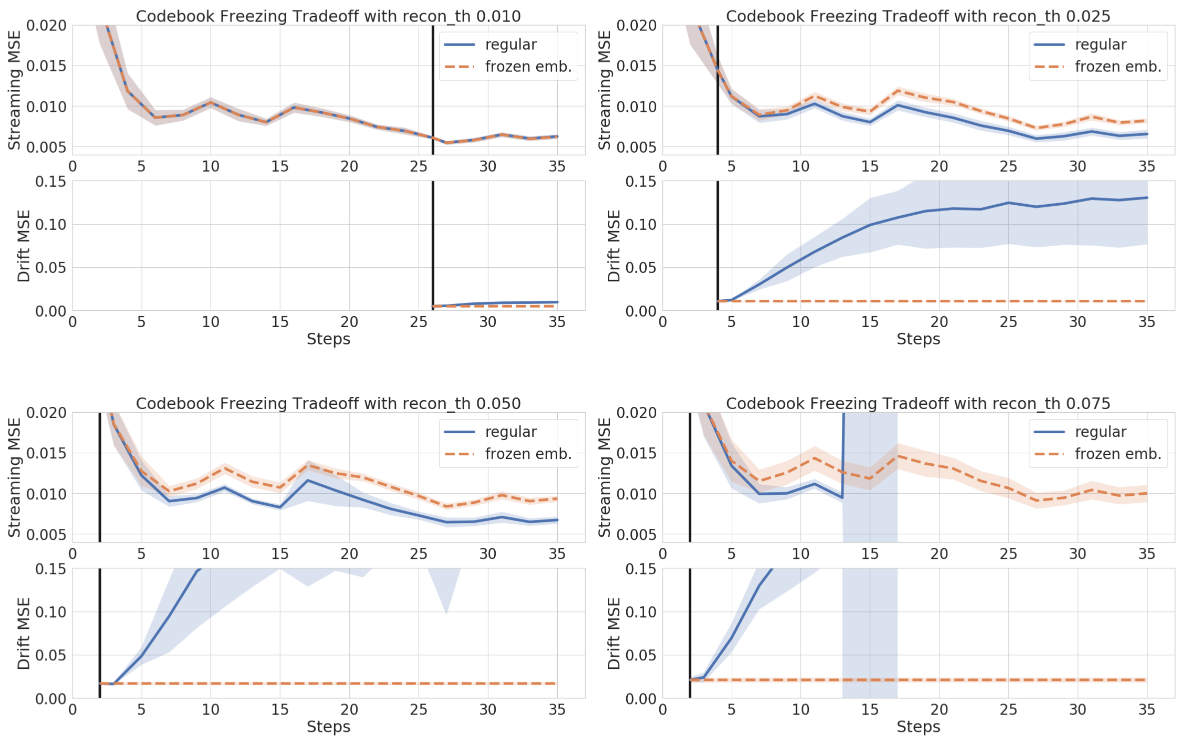

Figure 9. Visualization of results reported in Table C. We kept the scale of the y axis consistent across the four graphs for easy comparison.

D. Atari

Below are sample reconstructions used in the Atari experiments. For each game, reconstructions are in the first row and the original uncompressed samples in the second row. Reconstructions are $16\times$ smaller than the original RGB images.

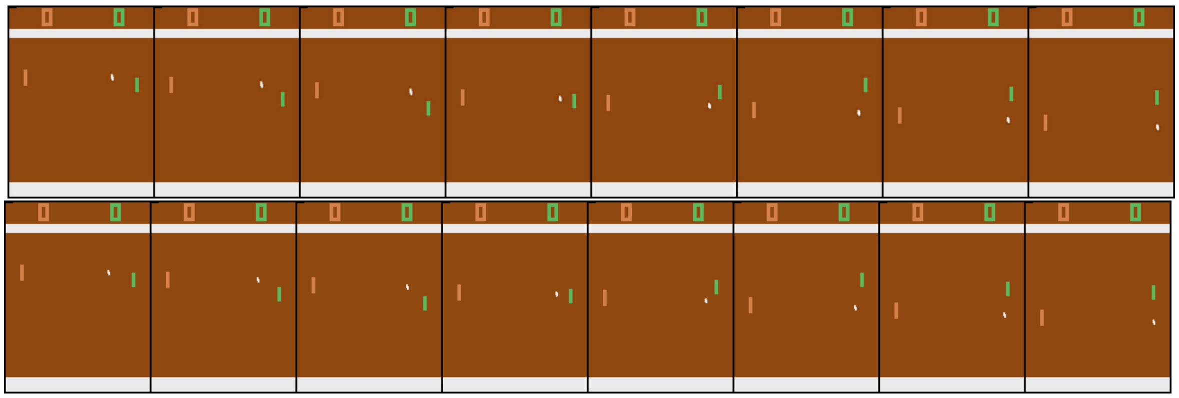

Figure 10. Pong

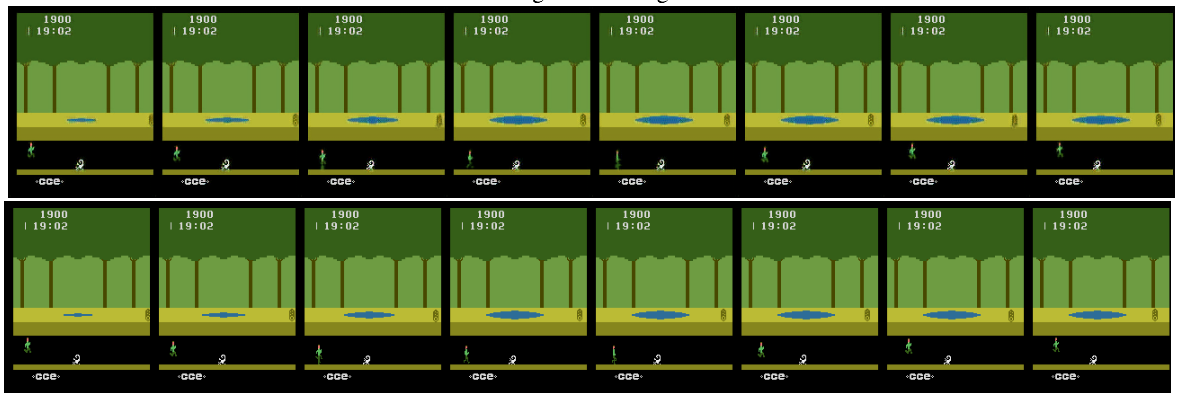

Figure 11. Pitfall

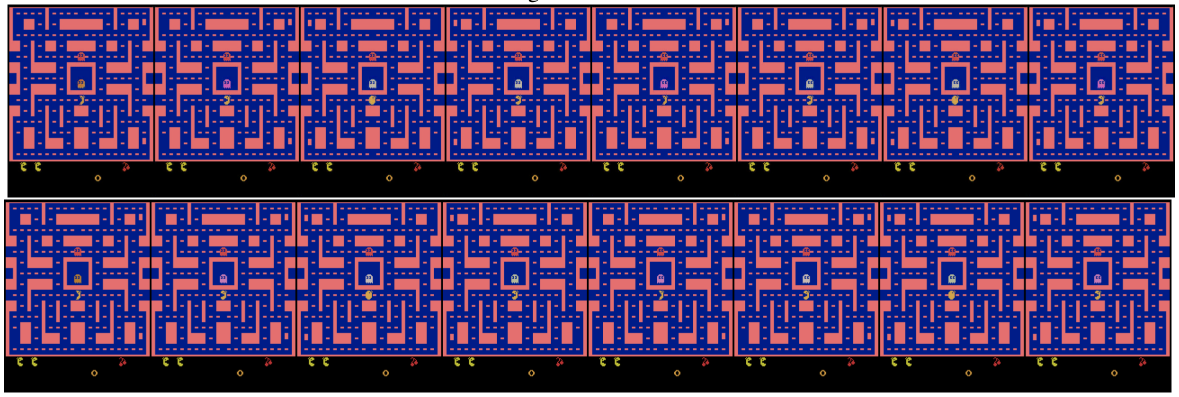

Figure 12. Ms Pacman

E. Lidar Samples

Here we show reveral lidar compressions (left) and their original counterpart (right). The compression rate is 32×. We note that unlike RGB images, raw lidar scans are stored using floating points. We calculate the compression rate from the (smaller) polar projection instead of the 3 channel cartesian representation.

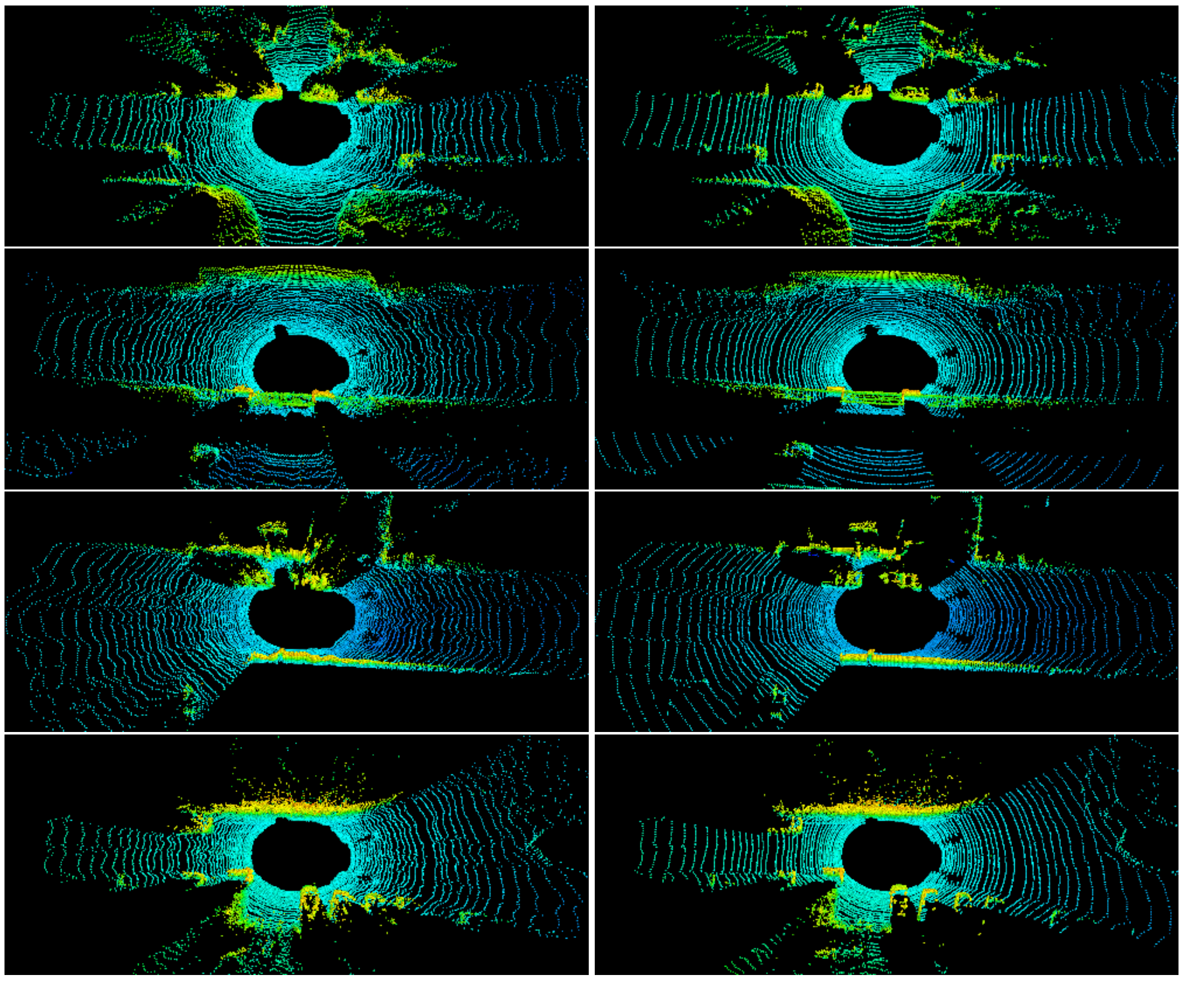

Figure 13. Lidar reconstruction (left) vs original (right)

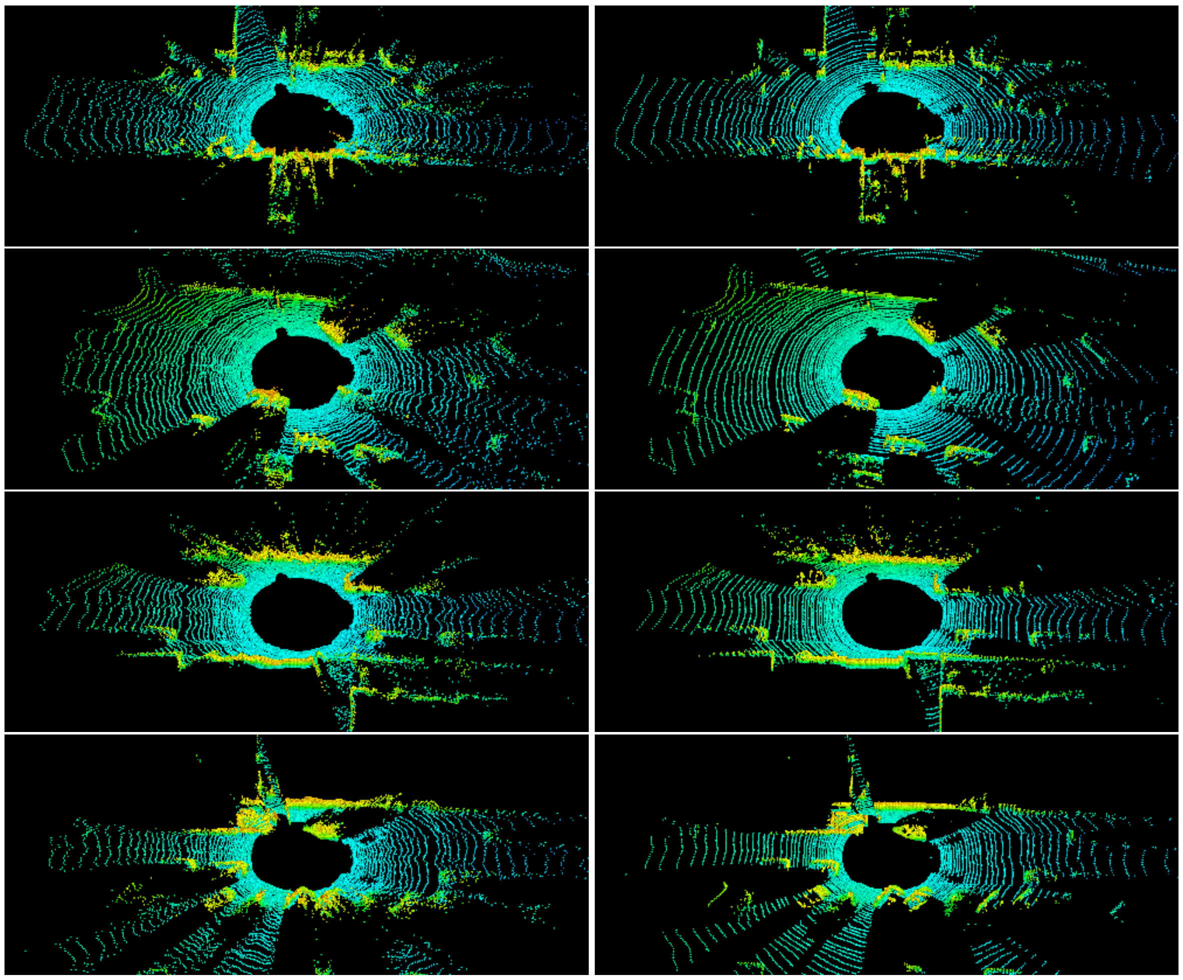

Figure 14. Lidar reconstruction (left) vs original (right)