A.1 Experimental Details for Section 3.2

For both the encoders and decoders, we use four fully-connected two-dimensional convolutional layers with 128 channels and a 3x3 kernel. We additionally use weight-normalization at each layer and PReLU activation units. Where relevant, we downsample with 2-stride convolutions at the third convolution, and upsample using a transposed convolution at the third convolution.

For training, we use the Adam optimizer with default learning settings and an initial learning rate of 5×10^{-4}. We exponentially anneal this learning rate to 1×10^{-5} during training.

Unlike stochastic posterior sampling, where we can train with continuous latent variables because discretization schemes cancel across distributions, deterministic posterior sampling requires discretization during training. Because the discrteization operation (i.e. rounding) is not differentiable, we adopt the popular technique of adding uniform noise during training, such that our discretized latent variable is defined by

$$\hat{z}^{(l)} = z^{(l)} + \epsilon, \quad \epsilon \sim \mathcal{U}\left(-\frac{\delta}{2}, \frac{\delta}{2}\right), \tag{1}$$

where δ is the uniform discretization bin and $\mathcal{U}$ represents a uniform distribution. In practice, we take $\delta = 1$.

A.2 Masked 3D Convolutions

For large k it becomes impractical to train using two-dimensional convolutions. Doing so typically necessitates a serial scheme across data partitions at a given latent variable. One approach to train our models in parallel is to use masked three-dimensional convolutions. We achieve this by expanding our downsampled data tensors into a $d\times Ck^2\times H\times W$ volume, where d is some auxiliary dimension.

In order to retain the causality constraints, we build our approach of two steps:

1. We use an *off-center convolution* of stride C to enforce the autoregressive structure *within* sub-blocks. We define this operation as one that concatenates a zero-tensor of dimension $d\times C\times H\times W$ to the data variable along the auxiliary dimension and then applies the convolution as described. The result of applying this convolution is a $f\times k^2\times H\times W$ tensor, where f are the output channels of the convolution.

2. We then apply repeated *masked* three-dimensional convolutions to the output of the the off-center convolution. To enforce the causality constraint between sub-blocks we apply a point-wise mask to the kernels prior to convolution. We define two types of masks: type 'A' and 'B'. We use B-type masks at all locations apart from the input, where use an A-type mask. We describe these masks in more detail below, and visualise them for a $3\times3\times3$ kernel in Figure 1.

For a three-dimensional kernel of depth d, height h and width w, consider the following masks that we apply as a point-wise multiplication to the kernel.

A-Type Mask

$$M_{d,h,w} = \begin{cases} 1 & \text{if } d \le \lfloor k/2 \rfloor \\ 0 & \text{else} \end{cases}. \tag{2}$$

B-Type Mask

$$M_{d,h,w} = \begin{cases} 1 & \text{if } d \le \lceil k/2 \rceil \\ 0 & \text{else} \end{cases}. \tag{3}$$

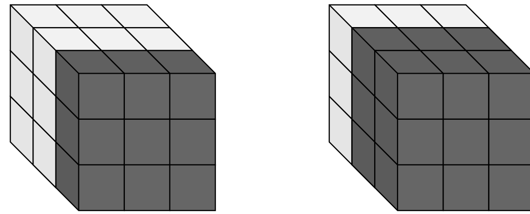

Figure 1. A $3\times3\times3$ kernel masking strategies. Mask type 'B' left; type 'A' right. Dark grey elements indicates zeros; light elements indicate ones. Masks are applied as point-wise multiplications to the kernel.

A.3 Visualisation and Coding Scheme for SHVC-ArIB

Variable Dependencies In Figure 2 and Figure 3, we illustrate the differences in the dependency structures in the priors and posteriors of SHVC and SHVC-ArIB. For ease of presentation, we do so using one latent variable (i.e. $L=1$) and assume $k=2$. We further assume $z^{(1)}$ is two-times smaller in spatial resolution that x but has the same number of channels, i.e. $C=3$.

Coding Schemes Here we use the above model specification as an example to illustrate the encoding and decoding processes of SHVC and SHVC-ArIB. Encoding and decoding algorithms for SHVC can be found in Algorithms 1 and 2. Encoding and decoding algorithms for SHVC-ArIB can be found in Algorithms 3 and 4. At the global level, the coding algorithm is consistent with that of Bit-Swap, and at the local level, the encoding of slices in latent and the data is conducted in the reverse order. Since the above model only involves one latent variable, the global level Bit-Swap degenerates to the original bb-ANS.

A.4 SHVC Architecture and Experimental Details

For both our encoder and decoder architectures, we use 8-layer Residual networks with PReLU activation units and weight-normalization. For CIFAR10 we additionally

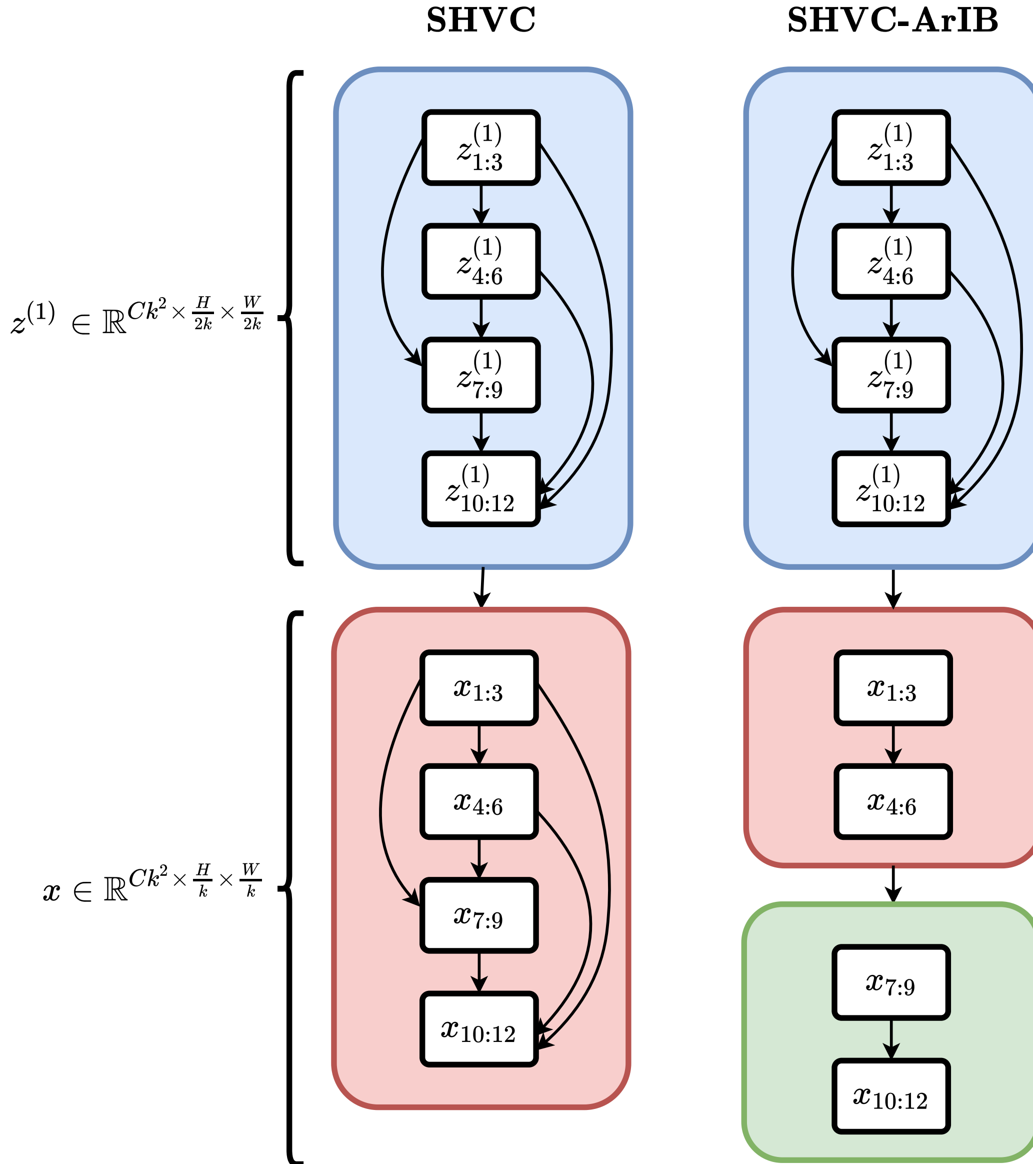

Figure 2. A comparison of the factorisation used in the priors of SHVC (left) and SHVC-ArIB (right). Variable groupings are represented by coloured blocks. Arrows indicate explicit dependencies. In SHVC-ArIB, there is no direct link between $z^{(1)}$ (blue) and $x_{7:12}$ (green).

use dropout layers between residual connections to prevent overfitting.

To highlight the effectiveness of our approach, we additionally train a small "Lite" model, which uses four fully-connected convolutional layers. Here we reduce the number of channels as we downsample the latent variables across layers. These are detailed as follows:

- $p(x|z^{(1)})$ uses 32 channels.
- $p(z^{(1)}|z^{(2)})$ uses 24 channels.
- $p(z^{(2)}|z^{(3)})$ uses 16 channels.
- $p(z^{(3)}|z^{(4)})$ uses 8 channels, if it exists.

For training, we use the Adam optimizer with default learning settings and an initial learning rate of 5×10^{-4}. We exponentially anneal this learning rate to 1×10^{-5} during training. We further use gradient-clipping to control for numerical stability.

We run all of our experiments on a single NVIDIA Tesla V100.

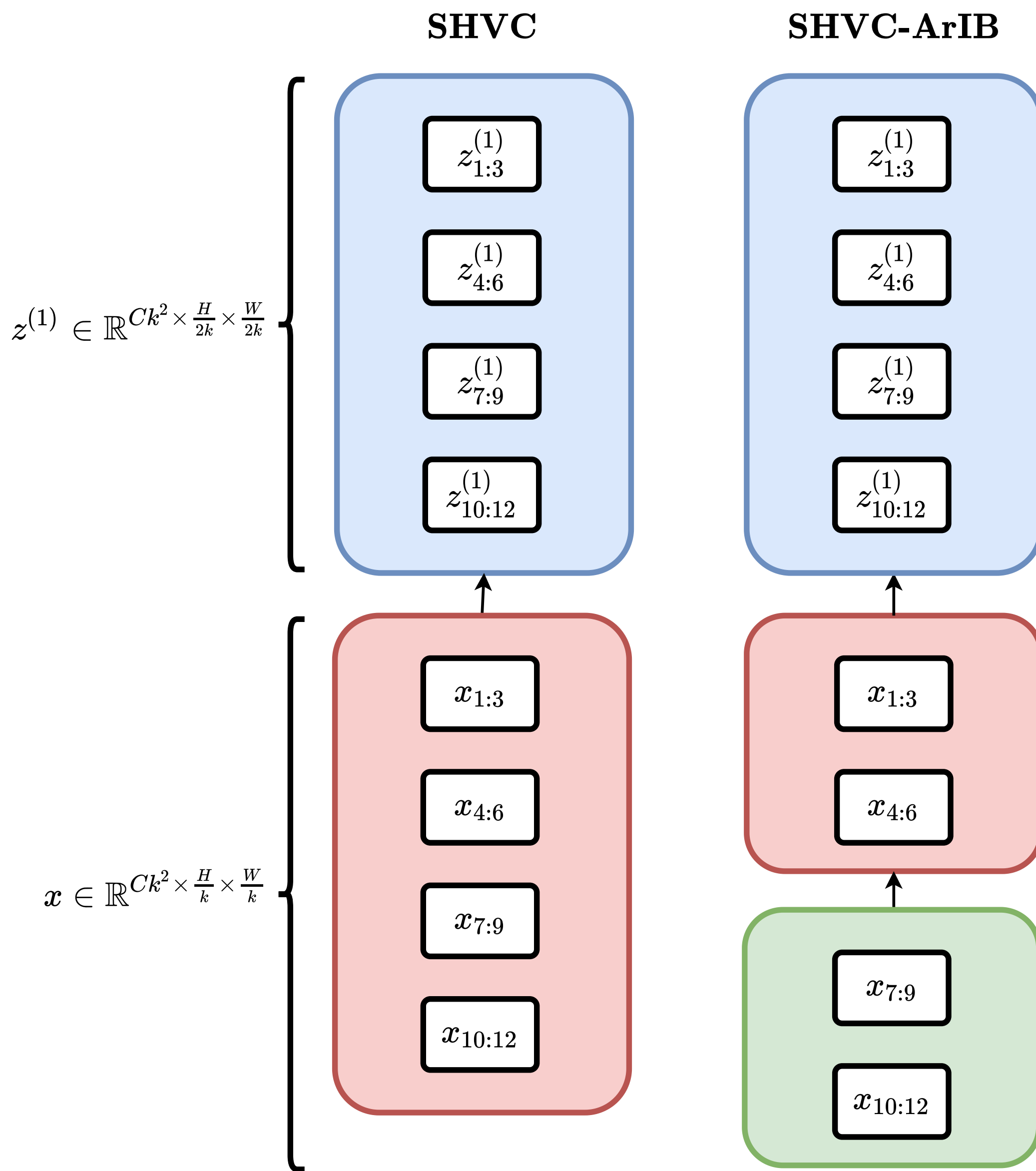

Figure 3. A comparison of the factorisation used in the posteriors of SHVC (left) and SHVC-ArIB (right). Variable groupings are represented by coloured blocks. Arrows indicate explicit dependencies. In SHVC-ArIB, there is no direct link between $z^{(1)}$ (blue) and $x_{7:12}$ (green).

Algorithm 1 SHVC Encoding

Input: data to compress x
Step 0: Get auxiliary initial bits c_0
Step 1: Decode $z^{(1)}$ with $q(z^{(1)}|x)$
Step 2: Encode x with $p(x|z^{(1)})$
  Encode x_{12} with $p(x_{12}|x_{1:11},z^{(1)})$
  Encode x_{11} with $p(x_{11}|x_{1:10},z^{(1)})$
  Encode x_{10} with $p(x_{10}|x_{1:9},z^{(1)})$
  ...
  Encode x_2 with $p(x_2|x_1,z^{(1)})$
  Encode x_1 with $p(x_1|z^{(1)})$
Step 3: Encode $z^{(1)}$ with $p(z^{(1)})$
  Encode $z^{(1)}_{12}$ with $p(z^{(1)}_{12}|z^{(1)}_{1:11})$
  Encode $z^{(1)}_{11}$ with $p(z^{(1)}_{11}|z^{(1)}_{1:10})$
  Encode $z^{(1)}_{10}$ with $p(z^{(1)}_{10}|z^{(1)}_{1:9})$
  ...
  Encode $z^{(1)}_{2}$ with $p(z^{(1)}_{2}|z^{(1)}_{1})$
  Encode $z^{(1)}_{1}$ with $p(z^{(1)}_{1})$
Output: final bit stream c

Algorithm 2 SHVC Decoding

Input: bit stream c
Step 1: Decode $z^{(1)}$ with $p(z^{(1)})$
  Decode $z^{(1)}_{1}$ with $p(z^{(1)}_{1})$
  Decode $z^{(1)}_{2}$ with $p(z^{(1)}_{2}|z^{(1)}_{1})$
  ...
  Decode $z^{(1)}_{10}$ with $p(z^{(1)}_{10}|z^{(1)}_{1:9})$
  Decode $z^{(1)}_{11}$ with $p(z^{(1)}_{11}|z^{(1)}_{1:10})$
  Decode $z^{(1)}_{12}$ with $p(z^{(1)}_{12}|z^{(1)}_{1:11})$
Step 2: Decode x with $p(x|z^{(1)})$
  Decode x_1 with $p(x_1|z^{(1)})$
  Decode x_2 with $p(x_2|x_1,z^{(1)})$
  ...
  Decode x_{10} with $p(x_{10}|x_{1:9},z^{(1)})$
  Decode x_{11} with $p(x_{11}|x_{1:10},z^{(1)})$
  Decode x_{12} with $p(x_{12}|x_{1:11},z^{(1)})$
Step 3: Encode $z^{(1)}$ with $q(z^{(1)}|x)$
Output: auxiliary initial bit stream c_0, data to decompress x

Algorithm 3 SHVC-ArIB Encoding

Input: data to compress x
Step 0: Get autoregressive initial bits by encoding $x_{7:12}$
  Encode x_{12} with $p(x_{12}|x_{1:11})$
  ...
  Encode x_7 with $p(x_7|x_{1:6})$
Step 1: Decode $z^{(1)}$ with $q(z^{(1)}|x_{1:6})$
Step 2: Encode $x_{1:6}$ with $p(x_{1:6}|z^{(1)})$
  Encode x_6 with $p(x_6|x_{1:5},z^{(1)})$
  ...
  Encode x_1 with $p(x_1|z^{(1)})$
Step 3: Encode $z^{(1)}$ with $p(z^{(1)})$
  Encode $z^{(1)}_{12}$ with $p(z^{(1)}_{12}|z^{(1)}_{1:11})$
  Encode $z^{(1)}_{11}$ with $p(z^{(1)}_{11}|z^{(1)}_{1:10})$
  Encode $z^{(1)}_{10}$ with $p(z^{(1)}_{10}|z^{(1)}_{1:9})$
  ...
  Encode $z^{(1)}_{2}$ with $p(z^{(1)}_{2}|z^{(1)}_{1})$
  Encode $z^{(1)}_{1}$ with $p(z^{(1)}_{1})$
Output: final bit stream c

Algorithm 4 SHVC-ArIB Decoding

Input: bit stream c
Step 1: Decode $z^{(1)}$ with $p(z^{(1)})$
  Decode $z^{(1)}_{1}$ with $p(z^{(1)}_{1})$
  Decode $z^{(1)}_{2}$ with $p(z^{(1)}_{2}|z^{(1)}_{1})$
  ...
  Decode $z^{(1)}_{10}$ with $p(z^{(1)}_{10}|z^{(1)}_{1:9})$
  Decode $z^{(1)}_{11}$ with $p(z^{(1)}_{11}|z^{(1)}_{1:10})$
  Decode $z^{(1)}_{12}$ with $p(z^{(1)}_{12}|z^{(1)}_{1:11})$
Step 2: Decode $x_{1:6}$ with $p(x_{1:6}|z^{(1)})$
  Decode x_1 with $p(x_1|z^{(1)})$
  ...
  Decode x_6 with $p(x_6|x_{1:5},z^{(1)})$
Step 3: Encode $z^{(1)}$ with $q(z^{(1)}|x_{1:6})$
Step 4: Decode $x_{7:12}$ with $p(x_{7:12}|x_{1:6})$
  Decode x_7 with $p(x_7|x_{1:6})$
  ...
  Decode x_{12} with $p(x_{12}|x_{1:11})$
Output: data to decompress x