TITLE: Verification Model Description for JPEG Pleno Learning-based Point Cloud Coding v4.0

SOURCE: WG1

EDITORS: André Guarda, Stuart Perry

PROJECT: JPEG Pleno

STATUS: Approved

REQUESTED ACTION: For Information

DISTRIBUTION: Public

Contact:
ISO/IEC JTC 1/SC 29/WG1 Convener - Prof. Touradj Ebrahimi
EPFL/STI/IEL/GR-EB, Station 11, CH-1015 Lausanne, Switzerland
Tel: +41 21 693 2606, Fax: +41 21 693 7600, E-mail: Touradj.Ebrahimi@epfl.ch
**Editorial Comments**

This is a living document that goes through iterations. Proposals for revisions of the text can be delivered to the editors André Guarda and Stuart Perry, by downloading this document, editing it using track changes and sending it to andre.guarda@lx.it.pt and stuart.perry@uts.edu.au

If you have interest in JPEG Pleno Point Cloud, please subscribe to the email reflector, via the following link: [http://jpeg-pointcloud-list.jpeg.org](http://jpeg-pointcloud-list.jpeg.org)
# Table of Contents

1. Executive Summary ........................................................................................................... 4  
2. Architecture and High-level Walkthrough Description ....................................................... 4  
3. Detailed Description of Geometry-related Modules .............................................................. 6  
   3.1 PC Block Partitioning/Merging .................................................................................. 6  
   3.2 Block Down/Up-sampling ....................................................................................... 7  
   3.3 DL-based Block Encoding and Decoding ................................................................ 7  
   3.4 DL-based Block Super Resolution ........................................................................ 10  
   3.5 Binarization Optimization ...................................................................................... 12  
4. Detailed Description of Colour-related Modules ................................................................. 12  
   4.1 Recolouring ............................................................................................................ 12  
   4.2 3D to 2D Projection and 2D to 3D Inverse Projection .............................................. 13  
   4.3 Image Encoding and Decoding ............................................................................... 14  
   4.4 Colour Super Resolution ....................................................................................... 14  
5. DL Model Training ............................................................................................................ 14  
   5.1 Geometry Coding Model Training ......................................................................... 14  
   5.2 Geometry SR Model Training ............................................................................... 16  
6. Coding Configurations ....................................................................................................... 17  
7. References ......................................................................................................................... 17
Verification Model Description for JPEG Pleno Learning-based Point Cloud Coding v4.0

January 26th, 2024

1 Executive Summary

This document describes the JPEG Pleno Point Cloud Coding [wg1/N100097] Verification Model (VM), consisting of a separate point cloud (PC) geometry and colour codec [wg1/M98031].

2 Architecture and High-level Walkthrough Description

The VM codec consists of two main stages, with the first compressing the PC geometry data, and the second compressing the PC colour data.

The overall architecture of the VM codec is presented in Figure 1.

![Figure 1 Overall architecture of the Verification Model codec.](image)

The coding walkthrough is described as follows:

- **Encoder:**
  - **PC Block Partitioning:** The voxelised PC geometry is divided into disjoint 3D blocks of a fixed target size, which are coded separately; the size of these block defines the random access granularity.
  - **Block Down-sampling:** Depending on the PC characteristics, each partitioned block may be down-sampled to a lower grid precision; this is typically
advantageous when the PC is sparse, since it increases the density of the blocks to be encoded.

○ **DL-based Block Encoding:** Each geometry block is encoded with an end-to-end DL coding model. This process can be compared to a typical transform coding approach, except that in this case a convolutional autoencoder (AE) is used to learn a non-linear transform. The transform generates a set of coefficients, referred to as the *latent representation*, which are then explicitly quantized by scaling with a defined real-valued quantization step (QS), followed by rounding, and finally entropy coded, generating a geometry bitstream.

○ **Binarization Optimization:** In order to provide adaptability to different PC densities, this module optimizes the binarization process that will be applied at the decoder. Since the output of the DL model decoder does not immediately correspond to a PC, but rather to the probabilities of voxels being occupied, this binarization process has the task to select the voxels and thus PC coordinates which will be ‘occupied’, based on the generated probabilities. The optimization process at the encoder produces a binarization parameter k, which corresponds to the number of occupied voxels that will be reconstructed at the decoder, thus it needs to be transmitted in the bitstream.

○ **DL-based Block Decoding:** Since geometry coding is a lossy process, in order to encode the PC colour information of the resulting decoded points, the geometry blocks are first decoded using the decoder counterpart of the DL-based block encoder mentioned before.

○ **PC Block Merging:** The decoded blocks are merged to reconstruct the full PC geometry.

○ **Recolouring:** The decoded geometry is recoloured with the original PC colour.

○ **3D to 2D Projection:** The recoloured PC is projected onto a 2D image.

○ **Image Encoding (JPEG AI VM):** The 2D image with the colour data is encoded using a DL-based image codec, such as the JPEG AI VM, generating a colour bitstream.

- **Decoder:**
  
  ○ **DL-based Block Decoding:** The geometry bitstream is decoded using the decoder counterpart of the previously mentioned DL-based block encoder, generating the geometry blocks.

  ○ **Block Up-sampling:** If down-sampling was performed at the encoder, each block is here up-sampled back to the original precision, without increasing the number of points, thus using a simple up-sampling solution.

  ○ **DL-based Block Super Resolution (optional):** This DL-based post-processing module can be used to apply super resolution (SR) to each block (since a simple up-sampling has already been performed in the Block Up-sampling module), in order to increase the number of points and, therefore, the density of the reconstructed PC at no rate cost.
PC Block Merging: The decoded blocks are merged to reconstruct the full PC geometry.

Image Decoding (JPEG AI VM): The colour bitstream is decoded, using the JPEG AI VM, generating a 2D image.

2D to 3D Inverse Projection: Using the already decoded geometry, the inverse projection operation is performed to obtain the decoded PC colour.

Colour Super Resolution: After performing geometry SR, the newly generated points need to be recoloured using a colour SR approach, generating the final decoded PC.

3 Detailed Description of Geometry-related Modules

Each of the codec architecture modules presented in Figure 1 is described here in more detail.

3.1 PC Block Partitioning/Merging

Before encoding, the PC is converted into a voxel-based 3D block representation, defining a regular structure that allows the use of convolutional neural networks (CNNs), similarly to image and video data. The geometry data is represented as a binary signal where a ‘1’ corresponds to an occupied voxel while a ‘0’ corresponds to an empty voxel.

However, due to the significant computational complexity overhead brought by this type of representation, a sparse tensor representation is considered instead. This sparse tensor representation only requires to explicitly represent the non-empty voxels, via their coordinates and corresponding values/features, while the remaining voxels are assumed to be empty. This sparse tensor representation allows a significant reduction in terms of required memory and computation time, since the empty voxels are not stored or used in the computations.

An example of the sparse tensor representation is shown in Figure 2.

![Voxel-based 3D Block Representation](image)

<table>
<thead>
<tr>
<th>Coordinates C</th>
<th>Features F</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 2 0</td>
<td>1</td>
</tr>
<tr>
<td>5 2 0</td>
<td>1</td>
</tr>
<tr>
<td>3 3 0</td>
<td>1</td>
</tr>
<tr>
<td>6 3 0</td>
<td>1</td>
</tr>
<tr>
<td>3 4 0</td>
<td>1</td>
</tr>
<tr>
<td>6 4 0</td>
<td>1</td>
</tr>
<tr>
<td>4 5 0</td>
<td>1</td>
</tr>
<tr>
<td>5 5 0</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 2 Example of sparse tensor representation.
Considering this new representation, a straightforward way to organize a PC is to divide it into disjoint blocks of a specific size, e.g., 64×64×64, which can then be coded separately with a DL geometry coding model. The position of each single 3D block is transmitted to the decoder.

At the decoder side, given the decoded blocks and their position, the full PC is reconstructed by merging the blocks accordingly.

### 3.2 Block Down/Up-sampling

This pair of modules is used when appropriate depending on the PC characteristics to reduce the PC coding precision (at encoder), allowing a more efficient compression, and then to restore the original precision (at decoder). This is achieved by simply scaling the input PC coordinates by a given sampling factor, followed by a rounding operation, which in turn induces a loss of points/information, but results in a denser surface (also larger voxels). At the decoder, the reconstructed PC coordinates are simply scaled back by the inverse sampling factor, which does not change the number of points or their relative positions.

This approach is particularly useful in two situations:

- **Sparse PCs coding**: Since the DL geometry coding model requires the conversion of the PC into a 3D block of binary voxels, the entire 3D space is represented including the empty voxels. When dealing with sparse PCs, the number of actual points in each block is much smaller than for dense PCs, meaning that the expended bits per input point (occupied voxels) tends to be significantly higher. In addition, the DL geometry coding model tends to have more difficulty to correctly reconstruct sparser surfaces. As such, by reducing the coding precision, the coding blocks are densified, giving an easier task to the DL geometry coding model, and bringing the coding rate to a more reasonable range.

- **Low rate coding**: It is common to experience some limitations when trying to train a DL geometry coding model to reach low rates, even for dense PCs. Furthermore, the quality of the reconstructed PCs at low rates can be considerably degraded. This approach provides a simple solution to reach low rates with fewer severe coding artifacts by down-sampling the PC and coding the blocks with DL geometry coding models trained for higher qualities rates.

### 3.3 DL-based Block Encoding and Decoding

This section presents the adopted DL-based PC geometry coding model acting at block-level. Based on successful CNN architectures for image coding [Ballé, 2018], the adopted end-to-end DL geometry coding model is presented in Figure 3. All convolution layers are sparse convolutions, implemented with Minkowski Engine v0.5.4 [Choy, 2019].

The full architecture can be divided into five main coding stages as follows:

- **Autoencoder**: The convolutional autoencoder (AE) transforms the input 3D block into a latent representation with lower dimensionality, in a way comparable to the transform coding stage in traditional image coding. This latent representation can be regarded as the coefficients of the transform, and consists of multiple feature maps, which number depends on the chosen number of filters for the convolutional layers. The AE consists of
a combination of 3D convolutional layers and Inception-Residual Blocks (IRB). The IRB is inspired in the Inception-ResNet [Szegedy, 2017], a popular neural network used for diverse image processing tasks; it contains several convolutional layers in parallel with different filter support sizes, which allow to extract different types of features from varying neighbouring contexts (from 5×5×5 to 1×1×1); in addition, a residual skip connection allows to propagate the features along the network, which also facilitates the training of deeper models. The number of filters starts from 32 in the first layer, and progressively increases to 128 at the final encoder layer, resulting in a rich latent representation. The AE contains a total of 2336160 trainable parameters, with 1208432 at the encoder side, and 1127728 at the decoder side.
- **AE Latents Coordinate Coding**: The coordinates ($C$) of the block latent representation generated by the AE are losslessly coded using octree coding, such as the Geometry-based Point Cloud Compression (G-PCC) standard [ISO, 2023].

- **Variational Autoencoder**: A variational autoencoder (VAE) is used to capture possible structure information still present in the block latent representation, which is then used as a hyperprior for the conditional entropy coding model; the mean-scale hyperprior as proposed in [Minnen, 2018] was adopted. This way, the entropy coding model parameters, consisting of the mean and scale of each of the Gaussians, can be more accurately estimated and adapted for each coded block. In this process, the VAE generates its own latent representation, which must also be coded and transmitted in the bitstream as additional side information to the decoder, so that the entropy coding model parameters can be replicated at the decoder. The VAE has 3 convolutional layers at the encoder, and a symmetric decoder with 2 identical branches, where the first one produces the means $\mu$ and the second one produces the scales (standard deviations) $\sigma$. The VAE decoder branch that produces the scales is quantized/integerized, meaning that all input, weights, biases, activations, and operations are integers, in order to guarantee bit exact behaviour [wg1/M102079]. The VAE contains a total of 2737152 trainable parameters.

- **VAE Latents Fixed Entropy Coding**: This module entropy codes the VAE latent representation. It uses a fixed entropy coding model for all blocks, which is learned during training. As all the components of the end-to-end DL geometry coding model are jointly trained, the additional side information rate is compensated by reducing the rate associated with the latents, thus optimizing the overall RD performance.

- **Quantization**: The AE latent representation is explicitly quantized before entropy coding. Considering a given quantization step ($QS$), which can be any positive real value, the latents are first scaled by $QS$. Then, the means $\mu$ obtained with the VAE are subtracted to obtain a residue latent representation which is then rounded to the closest
integer. This explicit quantization approach allows to fine tune the target rate at coding time for a single trained DL geometry coding model. At training time, an implicit quantization approach is considered (i.e., QS=1), and the rounding is replaced by a differentiable approximation, which consists in adding uniform noise to simulate the quantization error [Ballé, 2018].

- **AE Latents Conditional Entropy Coding:** The features (F) of the residue latent representation are entropy coded using a conditional entropy coding approach. It uses a Gaussian scale mixture conditioned on a hyperprior as the entropy coding model, with the scales \( \sigma \) obtained with the VAE. During training, the entropy of the latent representation is estimated according to the entropy coding model, which is then used for the rate-distortion (RD) optimization process. At coding time, a range encoder is used to create the block coding bitstream.

The total number of trainable parameters in the full DL geometry coding model (AE + VAE) is 5073312.

At the decoder side, each block is decoded with the DL geometry coding model shown in Figure 3. The “Side Info Bitstream” containing entropy coding related metadata is decoded to generate the entropy coding model parameters used for the current block, so that its “Bitstream” can finally be decoded.

### 3.4 DL-based Block Super Resolution

This section presents the optional DL-based Block Super Resolution module. It is important to notice that it receives the output of the Block Up-sampling module, which means that the PC is already in the original precision, although sparser. Its goal is to densify the PC, increasing the reconstructed quality at no rate cost; naturally, some complexity cost is involved. In practice, the DL geometry SR model expresses how a surface may be densified given a sparser surface, in this case within a PC block.

The DL-based Block SR module can offer significant RD performance gains, especially for originally dense and uniform PCs. However, this is not always the case, depending on the PC content characteristics (e.g., sparsity), as well as the reconstruction quality of the DL-based codec (whether it contains many coding artifacts or not). For this reason, the SR is an optional post-processing module which may be activated via the configuration used to run the software.
The DL geometry SR model architecture is based on the solution proposed in [Akhtar, 2020], and consists of a 3D CNN shaped as a U-net [Çiçek, 2016], as shown in Figure 4. All convolution layers are sparse convolutions, implemented with Minkowski Engine v0.5.4 [Choy, 2019].

The full architecture can be divided into two main processing stages as follows:

- **Contracting Path**: The first path of the U-net is responsible for extracting features at different scales. Similarly to the AE in the DL geometry coding model, it combines 3D down-sampling convolutional layers and IRB blocks, inspired in Inception-ResNet [Szegedy, 2017]. Compared to the DL geometry coding model, the IRB is much simpler and lighter, with fewer and smaller filter supports (maximum of $3 \times 3 \times 3$). On the other hand, the DL geometry SR model is a much deeper network, with many more convolutional layers and IRB’s. The number of filters/channels starts from 16 in the first layer, and progressively increases.
• **Expanding Path**: The second path now successively up-samples the features, but with an additional task of aggregating the multiscale features extracted by the contracting path. By considering the features obtained at different scales, this path is able to accurately predict the occupation of the voxels which were lost due to the down-sampling process performed at encoder. While regular up-sampling convolution layers do not change the number of points, the last up-sampling convolutional layer is a generative layer which is responsible for creating new points, depending on the kernel size \( K \). The kernel size was defined depending on the sampling factor, namely \( K = 3 \times 3 \times 3 \) for sampling factor 2, and \( K = 5 \times 5 \times 5 \) for sampling factor 4.

The total number of trainable parameters in the full DL geometry SR model is 7253817.

### 3.5 Binarization Optimization

The output of the DL-based geometry coding model consists of values in the interval \([0, 1]\) for each voxel, where each value represents the probability of the given voxel being occupied. As such, it is necessary to transform these probabilities into binary values which can directly correspond to the final reconstructed points. A so-called **optimized Top-k** binarization approach was adopted for selecting the occupied voxels, in which only the \( k \) voxels with the largest probabilities are selected as points, with \( k \) being defined as:

\[
k_{\text{Codec}} = N_{\text{input}} \times \beta,
\]

where \( N_{\text{input}} \) is the number of input points of the original block (known), and \( B \) is a factor selected at the encoder. This \( B \) factor is optimized at the encoder via an exhaustive linear search over a pre-defined range of meaningful values, with the value which results in the best reconstructed quality (using a geometry quality metric such as PSNR-D1 or PSNR-D2) being selected; naturally, the best value of \( k_{\text{Codec}} \) needs to be coded in the bitstream to be available to the decoder.

Similarly, the output of the DL geometry SR model is a block of occupied voxel probabilities, thus also requiring some binarization process. The same optimized Top-\( k \) approach used after the DL geometry coding model is applied; however, this requires SR optimization during the encoding process, i.e. performing SR at the encoder side, which can considerably increase the encoding time and complexity. A binarization parameter \( k_{\text{SR}} \), computed as in Equation (1), needs to be transmitted in the bitstream together with the \( k_{\text{Codec}} \) parameter (*Binarization Parameters* in Figure 1).

### 4 Detailed Description of Colour-related Modules

#### 4.1 Recolouring

Before to encoding the colour, the geometry is first encoded and decoded. As this is a lossy coding process, the number of points in the decoded PC and their positions may change regarding the original PC. To avoid coding and transmitting colour information of points that will no longer exist at the decoder, a recolouring process is used to transfer the colour from the original PC to the decoded geometry (at the encoder) as follows:

- For collocated points, simply copy the colour from the original PC.
4.2 3D to 2D Projection and 2D to 3D Inverse Projection

The 3D to 2D projection used by the Video-based Point Cloud Compression (V-PCC) standard [ISO, 2021] was adopted for the VM codec, with a few modifications to remove any colour dependence during the patch refinement stage in V-PCC; this is essential to enable the inverse projection at the decoder without any need for side information, which would require additional rate. Regarding the V-PCC projection parameters, the smoothed push-pull algorithm [WG7/N00100] was selected for the background filling of the projected 2D image, and the 2D image width was limited to the maximum size of the bounding box of the PC geometry.

Since points with different coordinates may be potentially projected onto the same pixel, V-PCC generates two projected images, named near and far layers. The two layers can often be very similar, depending on the PC geometry complexity, since the points in simpler surfaces may be projected on both images. To avoid coding redundant information, the far layer is coded differentially as follows:

1. Compute the difference (residue) between far and near layers.
2. Trim the far layer residue image to consider only the smallest rectangle that includes all non-zero residue pixels.
3. Represent the residue image as 8-bit unsigned integers, via scaling by a factor of 2 and a shift of 128 to reach the range \([0, 255]\).

Naturally, the size and position of the trimmed residue image (inside the full image) are transmitted to the decoder.

Figure 5 shows an example of the V-PCC projection and the resulting far layer residual image, for the Iguana PC.
4.3 Image Encoding and Decoding
The 2D images obtained with the V-PCC projection are coded using the JPEG AI VM. A description of this module may be found in [WG1/N100332].

4.4 Colour Super Resolution
The colour SR module is used to determine the colour of the new points generated by the geometry SR module (DL-based Block Super Resolution). For this purpose, the same interpolation approach used in the previous Recolouring module is adopted, namely the colour of the newly created points is determined via a Radial Basis Function (RBF) interpolation with linear kernel, considering the 20 nearest neighbours.

5 DL Model Training
This section describes the important training processes for the DL coding and SR models (for geometry only).

5.1 Geometry Coding Model Training
In order to achieve efficient compression performance, the DL geometry coding model from Figure 3 was trained by minimizing a loss function that considers both the distortion of each decoded block, compared to the input block, as well as its estimated coding rate. For this purpose, the loss function follows a traditional formulation involving a Lagrangian multiplier, $\lambda$, and is given by:

$$\text{Loss Function} = \text{Distortion} + \lambda \times \text{Coding rate}.$$  \hspace{1cm} (2)

DL-based codecs typically require training a different DL geometry coding model for each target RD point, which is accomplished by varying the $\lambda$ parameter in Equation (2). A total of 5 models were trained, using $\lambda = 0.0025, 0.005, 0.01, 0.025,$ and $0.05$. Models were trained sequentially, from the smallest to the largest $\lambda$ (i.e., highest to lowest rate) [Quach, 2020]. This means that only the first model (highest rate) was trained with no particular initialization, whereas each subsequent DL geometry coding model was initialized with the weights of the previous one. This approach allows for a significant reduction in the total training time of the remaining models since their training is, in practice, a fine-tuning for the next target RD trade-off. Furthermore, as each subsequent DL geometry coding model becomes more and more refined, this sequential approach can also offer a better RD performance for the latter models, when compared to the regular approach of training each model independently, i.e., without initialization.

Training Distortion Metric
As described in Section 3.1, a voxel-based sparse tensor representation was adopted to process the PC geometry. Thus, for the DL geometry coding model the input data is a block of binary voxels, and the decoded data represents a probability score between '0' and '1' for each voxel, i.e. the probability of each voxel being occupied. While at encoding and decoding time binarization is eventually applied, it cannot be performed during training as this is not a differentiable operation. This means that the geometry distortion metrics used for testing (D1, D2) cannot be used for training since they require binarization.
Considering this, the geometry distortion is measured as the average voxel level distortion, computed as a binary classification error using the so-called Focal Loss (FL), defined as follows:

\[
FL(v, u) = \begin{cases} 
-\alpha(1 - v)^\gamma \log v, & u = 1 \\
-(1 - \alpha)v^\gamma \log(1 - v), & u = 0 
\end{cases}
\]

where \( u \) is the original voxel binary value and \( v \) is the corresponding decoded voxel probability value. A weight parameter, \( \alpha \), is used to control the class imbalance effect since the number of ‘0’ valued voxels in a block is vastly superior to the number of ‘1’ valued voxels. The parameter \( \gamma \) allows increasing the importance of correcting misclassified voxels in relation to improving the classification score of already correct voxels. For the used models, the values \( \alpha = 0.5 \) and \( \gamma = 2 \) for these two parameters were found to be appropriate.

**Training Coding Rate**

The coding rate is estimated during training as the entropy of the AE and VAE latent representations, considering the computed conditional and fixed entropy coding models, respectively.

**Trained Models**

The DL geometry coding model is trained using a selection of the static PCs listed in the JPEG Pleno PCC Common Training and Testing Conditions (CTTC) [wg1/N100112]. As detailed in Table 1, the selected PCs were down-sampled to a lower precision (if necessary) according to their sparsity, and then partitioned into blocks of size 64×64×64, as described in Section 3.1. The blocks with less than 500 ‘occupied’ voxels have been removed to avoid the negative effect on the training due to the increased class imbalance caused by such low point count blocks. In total, 35303 blocks were used for training and 3421 blocks for validation.

The PCs were split into training and validation sets, with the validation set being used for early stopping of the training process, in order to prevent overfitting. For early stopping, a patience of 25 epochs with a tolerance of 0.1% was defined, meaning that the training only stopped when the validation loss did not decrease below 0.1% of the previous minimum after 25 epochs.

### Table 1 - Dataset for training and validation of the DL geometry coding model.

<table>
<thead>
<tr>
<th>Point Cloud</th>
<th>Frame</th>
<th>Original Precision</th>
<th>Original Points</th>
<th>Training Precision</th>
<th>Training Points</th>
<th>Blocks (64³)</th>
<th>Blocks (128³)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loot</td>
<td>1200</td>
<td>10</td>
<td>805285</td>
<td>10</td>
<td>805285</td>
<td>192</td>
<td>49</td>
</tr>
<tr>
<td>Redandblack</td>
<td>1550</td>
<td>10</td>
<td>757691</td>
<td>10</td>
<td>757691</td>
<td>166</td>
<td>46</td>
</tr>
<tr>
<td>Andrew10</td>
<td>1</td>
<td>10</td>
<td>1276312</td>
<td>10</td>
<td>1276312</td>
<td>224</td>
<td>68</td>
</tr>
<tr>
<td>David10</td>
<td>1</td>
<td>10</td>
<td>1492780</td>
<td>10</td>
<td>1492780</td>
<td>277</td>
<td>81</td>
</tr>
<tr>
<td>Sarah10</td>
<td>1</td>
<td>10</td>
<td>1355867</td>
<td>10</td>
<td>1355867</td>
<td>258</td>
<td>75</td>
</tr>
<tr>
<td>The20sMaria</td>
<td>600</td>
<td>-</td>
<td>10383094</td>
<td>11</td>
<td>3681165</td>
<td>669</td>
<td>175</td>
</tr>
<tr>
<td>UlliWegner</td>
<td>1400</td>
<td>-</td>
<td>879709</td>
<td>10</td>
<td>537042</td>
<td>150</td>
<td>39</td>
</tr>
<tr>
<td>Basketball player</td>
<td>200</td>
<td>11</td>
<td>2925514</td>
<td>11</td>
<td>2925514</td>
<td>682</td>
<td>184</td>
</tr>
</tbody>
</table>
The learning-rate was initially set as $10^{-4}$, and reduced to $10^{-5}$ when the validation loss does not decrease below 0.1% of the previous minimum after 10 epochs.

Implementation and training were done in PyTorch version 1.12, using the CompressAI library version 1.2 [Bégaint, 2020] for entropy coding. For training, the Adam algorithm [Kingma, 2015] was used with minibatches of 8 blocks.

### 5.2 Geometry SR Model Training

The training of the DL geometry SR model did not follow the same approach as for the coding model due to its different purpose. Each block was first down and up-sampled (by the target scaling factor) using the down-sampling and up-sampling approaches described in Section 3.2; this process allows obtaining a sparser block although at the same original precision. The SR training data were thus pairs of sparse and corresponding original blocks, where the former served as the input to the DL geometry SR model, and the later served as the ground truth when measuring the SR training loss. Note that, just for training purposes, there was no coding involved.

Given that SR is a post-processing module which has no impact on the coding rate, the DL geometry SR model was trained simply considering the distortion between the mentioned blocks, computed using the loss function in Equation (3), with the same metrics and parameter values described in the previous section.

At testing time, a single DL geometry SR model is used for all the rates, unlike the DL geometry coding model. As such, the only dependency is the sampling factor, with two DL geometry SR models being trained in total, one considering a sampling factor of 2 and
another considering a sampling factor of 4. For sampling factor 2, the training PC dataset was divided into blocks of size $64 \times 64 \times 64$, just as for the DL geometry coding model; however, for sampling factor 4, the training PC dataset was divided into blocks of size $128 \times 128 \times 128$ instead, resulting in a total of 9572 blocks for training and 993 blocks for validation.

For early stopping, a patience of 25 epochs with a tolerance of 0.1% was defined, meaning that the training only stopped when the validation loss did not decrease below 0.1% of the previous minimum after 25 epochs.

The learning-rate was initially set as $10^{-4}$, and reduced to $10^{-5}$ when the validation loss does not decrease below 0.1% of the previous minimum after 10 epochs.

Implementation and training were done in PyTorch version 1.12. For training, the Adam algorithm [Kingma, 2015] was used with minibatches of 8 and 4 blocks for sampling factor of 2 and 4, respectively.

6 Coding Configurations

In DL-based coding solutions, it is typical to train a DL geometry coding model for a specific target RD trade-off. However, such approach becomes impractical when aiming to achieve a given bitrate at testing time, without the possibility of training new models. As such, the VM codec avoids this issue by allowing a more flexible coding configuration at testing time, containing several parameters for geometry coding, namely:

- **DL geometry coding model**: Five DL geometry coding models are available, which were trained for different RD trade-offs, spanning a wide range of rates.
- **Sampling factor**: The sampling factor can be defined to allow reaching lower rates (by increasing its value) or to improve coding performance for sparser PCs.
- **Super Resolution**: The SR module can be activated optionally, with the goal to improve the reconstruction quality when adopting a sampling factor larger than 1.
- **Coding block size**: The size of the 3D block coding units can be selected, allowing not only a finetuning of the rate, but also a trade-off between performance and random access granularity.
- **Quantization step**: The quantization step parameter applied to the latents can be used for finetuning the target rate for each PC after selecting a specific DL geometry coding model.

Furthermore, for colour coding, the desired image coding rate may be selected with the JPEG AI VM. This allows the user to control the balance between geometry and colour rates for any PC.

Considering these coding parameters and the desired target rates, it is possible to select the coding configurations which allow reaching the target rates for each PC.

7 References


