ISO/IEC JTC1/SC29/WG1 N100367
97th Meeting – Online – 24-28 October 2022

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE DE NORMALISATION

ISO/IEC JTC 1/SC 29/WG1
(ITU-T SG16)

Coding of Still Pictures

JBIG
Joint Bi-level Image Experts Group

JPEG
Joint Photographic Experts Group

TITLE: Verification Model Description for JPEG Pleno Learning-based Point Cloud Coding v1.0

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

ISO/IEC JTC1/SC29/WG1 **N100367**

97th Meeting – Online – 24-28 October 2022

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

ISO/IEC JTC1/SC29/WG1 N100367
97th Meeting – Online – 24-28 October 2022

Table of Contents

ISO/IEC JTC1/SC29/WG1 N100367
97th Meeting – Online – 24-28 October 2022

Verification Model Description for JPEG Pleno Learning-based Point Cloud Coding v1.0

October 28th, 2022

1 Executive Summary

This document describes the JPEG Pleno Point Cloud Coding [wg1/N100097] Verification Model (VM), consisting of a deep learning (DL)-based joint point cloud (PC) geometry and colour codec [wg1/M96005].

2 Architecture and High-level Description

The VM codec consists of two main DL-based neural network modules, one focusing on compressing the PC geometry and colour data, and the other (optional) focusing on post-processing with the goal of performing up-sampling/super-resolution to increase the quality at no rate cost. The VM codec offers a joint geometry and colour coding approach, in which the same DL model processes geometry and colour simultaneously.

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

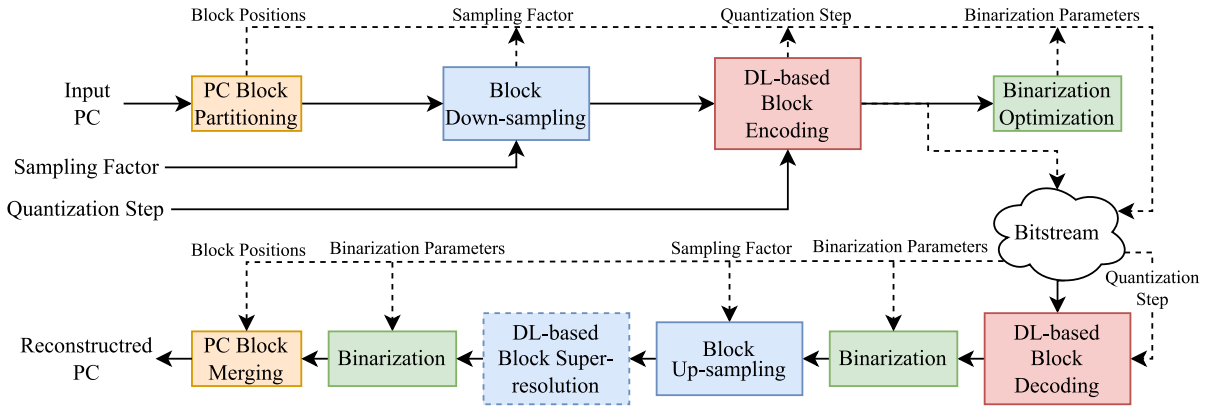


Figure 1 Overall architecture of the Verification Model codec.

The various modules are briefly described as follows:

- **Encoder:**
    - **PC Block Partitioning:** The voxelized PC 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 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.
- **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.

- **Decoder:**
  - **DL-based Block Decoding:** Blocks are decoded using the decoder counterpart of the DL-based block encoder mentioned before.
  - **Binarization:** The decoded voxel probabilities for each block are binarized to reconstruct the PC coordinates using the binarization parameter value resulting from the optimization performed at the encoder.
  - **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 basic grid 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.
  - **Binarization:** After the DL-based Block Super-resolution, it is necessary to once again apply the binarization process, since occupied voxel probabilities are created by the SR model.
  - **PC Block Merging:** The decoded blocks are merged to reconstruct the full PC

3 Detailed Module Description

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 input PC contains not only the geometry (3D coordinates) but also the colour attributes, typically in RGB colour space. As such, each voxel consists of four channels: the first channel represents the geometry information as a binary signal, where a '1' corresponds to an occupied voxel while a '0' corresponds to an empty voxel, and the remaining three channels represent the 8-bit RGB colour information, one channel for each component. In order for geometry and colour values to be in the same range, the RGB colour values are scaled from the original [0, 255] range to [0, 1]. For empty voxels, the colour values are assumed to be zero.

An example of the voxel-based representation is shown in Figure 2.

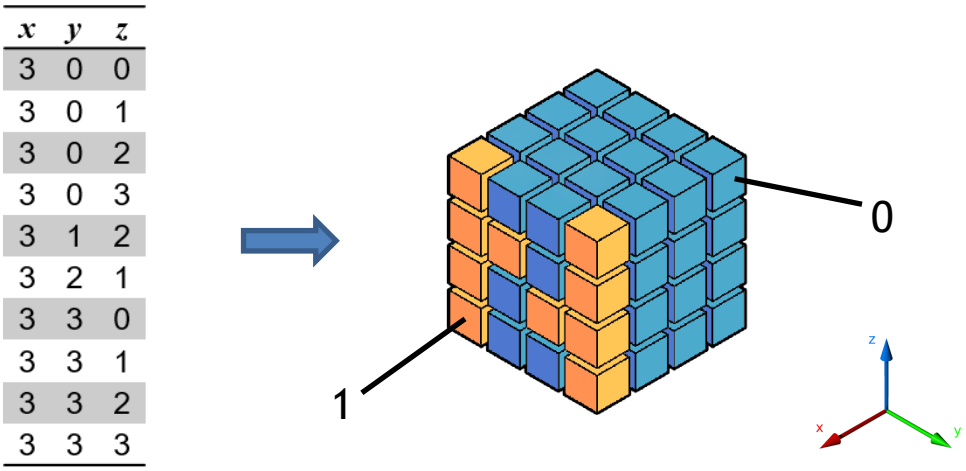


Figure 2 Example of conversion from 3D PC coordinates to a 3D block of binary voxels.

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 coding model. The position of each single 3D block is transmitted to the decoder (*Block Positioning* in Figure 1).

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 in 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

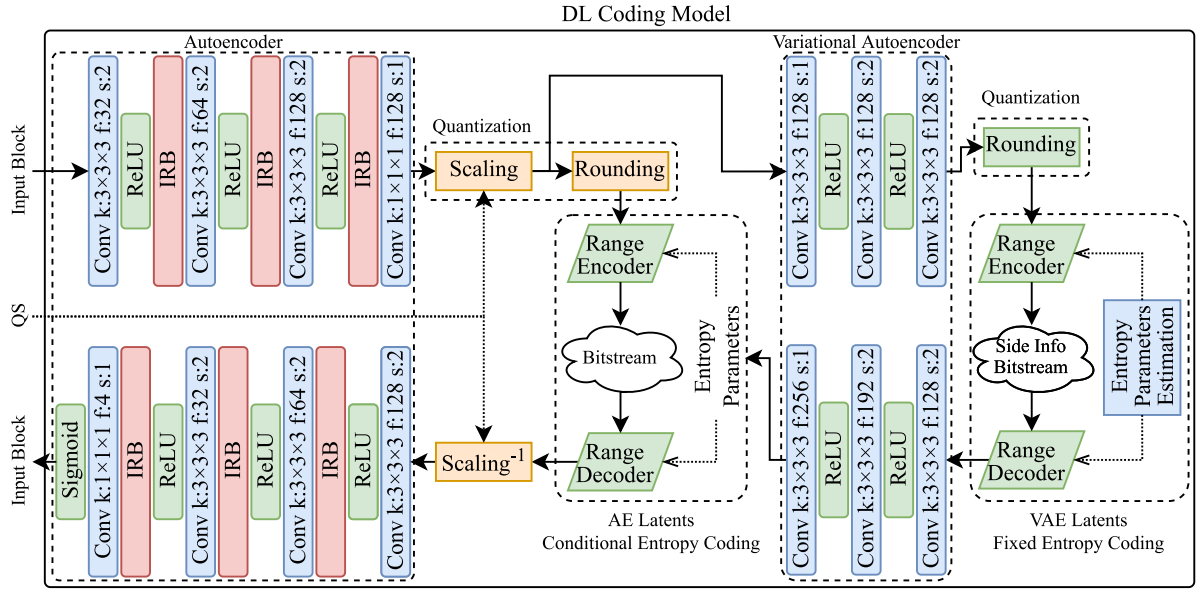
the decoder, the reconstructed PC coordinates are simply scaled back by the inverse sampling factor, which does not change the number of points or their relative positions.

This approach is particularly useful in two situations:

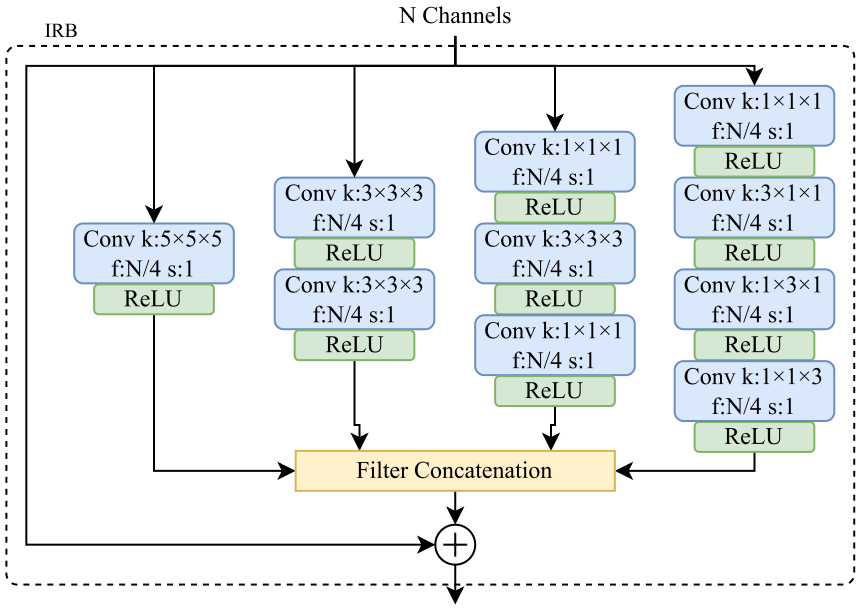
- **Sparse PCs coding:** Since the DL coding model requires the conversion of the PC into a 3D block of binary voxels, the entire 3D space is represented including the empty voxels. When dealing with sparse PCs, the number of actual points in each block is much smaller than for dense PCs, meaning that the expended bits per input point (occupied voxels) tends to be significantly higher. In addition, the DL 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 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 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 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 coding model is presented in Figure 3.



(a)



(b)

Figure 3 End-to-end DL coding model architecture: (a) Overall DL coding model; (b) Detail of the IRB block.

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 2844704 trainable parameters, with 1211024 at the encoder side, and 1633680 at the decoder side.

- **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, and 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 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:** A conditional entropy coding approach is used to entropy code the block latent representation. It uses a Gaussian mixture model conditioned on a hyperprior as the entropy coding model. 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.

- **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 a similar but simpler design to the AE, with only 3 convolutional layers at the encoder and another 3 at the decoder. The VAE contains a total of 3760960 trainable parameters.

- **VAE Latents Fixed Entropy Coding:** Similar to the conditional entropy coding, this module entropy codes the VAE latent representation. However, it uses a fixed entropy coding model for all blocks, which is learned during training, instead of an adaptive one for each block. As all the components of the end-to-end DL 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.

The total number of trainable parameters in the full DL coding model (AE + VAE) is 6605664.

At the decoder side, each block is decoded with the DL 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 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 SR model architecture is based on the solution proposed in [Akhtar, 2020], and consists on a 3D CNN shaped as a U-net [Çiçek, 2016], as shown in Figure 4.

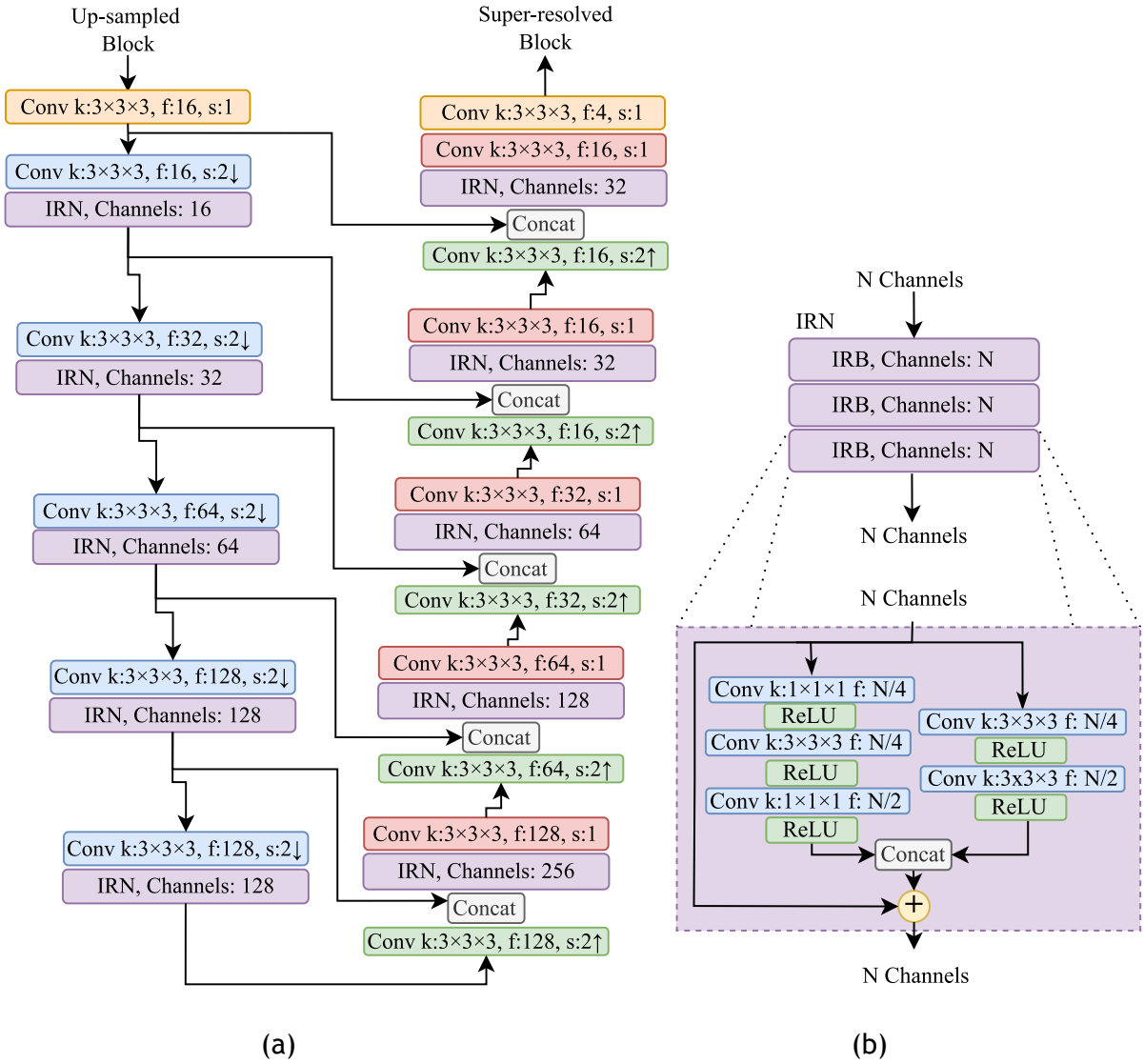


Figure 4 DL SR model architecture: (a) Overall SR model; (b) Detail of the IRN block.

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 coding model, it combines 3D down-sampling convolutional layers and IRB blocks, inspired in Inception-ResNet [Szegedy, 2017]. Compared to the DL coding model, the IRB is much simpler and lighter, with fewer and smaller filter supports (maximum of 3×3×3). On the other hand, the DL 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.

The total number of trainable parameters in the full DL SR model is 7291488.

3.5 Binarization

The geometry output of the DL-based decoder 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:

$$k_{Codec} = N_{input} \times \beta, \quad (1)$$

where N_{input} is the number of input points of the original block (known), and β is a factor selected at the encoder. This β factor is optimized at the encoder by determining the value which results in the best reconstructed quality, using a combination of geometry and colour quality metrics such as PSNR-D1 or PSNR-D2 and PSNR-YUV or PSNR-RGB, respectively; naturally, the best value of k_{Codec} needs to be coded in the bitstream to be available to the decoder.

Furthermore, the occupation of each octant of the block is determined and transmitted to the decoder so that only voxels inside originally occupied octants can be selected as occupied voxels.

Similarly to the DL coding model, the output of the DL SR model is a block of occupied voxel probabilities, thus also requiring some binarization process. To maximize the RD performance, two binarization approaches may be selected with different implications on the encoder complexity:

- The same optimized Top-*k* approach used after the DL coding model may be 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.
- Alternatively, the same binarization multiplying factor β determined for the DL coding model optimization can be applied, at no increased encoding complexity cost. While it can still bring a good RD performance, notably for dense PCs, this approach is not optimal.

In both these approaches, only a binarization parameter k_{SR}, computed as in Equation (1), needs to be transmitted in the bitstream together with the k_{Codec} parameter, as well as the octant occupation code (*Binarization Parameters* in Figure 1).

4 DL Model Training

This section describes the important training processes for the DL coding and SR models.

4.1 Coding Model Training

In order to achieve efficient compression performance, the DL 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, λ, and is given by:

$$Loss\ Function = Distortion + \lambda \times Coding\ rate. \quad (2)$$

DL-based codecs typically require training a different DL coding model for each target RD point, which is accomplished by varying the λ parameter in Equation (2). A total of 6 models were trained, using λ = 0.000125, 0.00025, 0.0005, 0.001, 0.002, and 0.004. Models were trained sequentially, from the smallest to the largest λ (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 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 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

Being a joint PC geometry and colour codec, the distortion of both geometry and colour information is measured, with the total distortion being given by:

$$Total\ Distortion = (1-\omega) \times Distortion_{Geometry} + \omega \times Distortion_{Colour}\ , \quad (3)$$

where ω is a weight that leverages the importance of colour over the geometry. Although more intensive studies may be performed on the best weight, ω=0.5 has been assumed.

As described in Section 3.1, a voxel-based representation was adopted to process the PCs. Thus, regarding geometry, for the DL 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 ($Distortion_{Geometry}$) is measured as the average voxel level distortion, computed as a binary classification error using the so-called *Focal Loss* (FL) [Lin, 2017], defined as follows:

$$FL(v,u) = \begin{cases} -\alpha(1-v)^{\gamma}\log v\,, & u=1 \\ -(1-\alpha)v^{\gamma}\log(1-v)\,, & u=0 \end{cases}, \quad (4)$$

where u is the original voxel binary value and v is the corresponding decoded voxel probability value. A weight parameter, α, 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 γ 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 α=0.7 and γ=2 for these two parameters were found to be appropriate.

As previously stated, the distortion function must be differentiable and thus the colour distortion metrics used for testing (MSE-Y, MSE-RGB) cannot be used. Thus, for the colour distortion ($Distortion_{Colour}$), a voxel-wise mean squared error (MSE) was defined, as follows:

$$Distortion_{Colour} = \frac{1}{N_{input}} \sum_{i \in N_{input}} \frac{(R_i - R_i')^2 + (G_i - G_i')^2 + (B_i - B_i')^2}{3}, \qquad (5)$$

where R_i, G_i, B_i are the colour values of the occupied voxel *i* in the input block, R_i', G_i', B_i' are the colour values of the collocated voxel in the decoded block, and N_{input} is the number of occupied voxels in the input block.

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. Since the latent representation contains both the compressed geometry and colour representation without separation between them, only the total (geometry + colour) rate is considered.

Trained Models

The DL 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, 35861 blocks were used for training and 3822 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 5 epochs was defined, meaning that the training only stopped when the validation loss did not decrease further after 5 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 a learning rate of 10^{-4} and minibatches of 16 blocks.

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

	Point Cloud	Frame	Original Precision	Original Points	Training Precision	Training Points	Blocks (64^3)
Training	Loot	1200	10	805285	10	805285	192
	Redandblack	1550	10	757691	10	757691	166
	Soldier	690	10	1089091	10	1089091	235
	Thaidancer	viewdep	12	3130215	11	1007956	222
	Andrew10	1	10	1276312	10	1276312	224
	David10	1	10	1492780	10	1492780	277
	Sarah10	1	10	1355867	10	1355867	258

	The20sMaria	600	Not voxelized	10383094	11	3681165	669
	UlliWegner	1400	Not voxelized	879709	10	537042	150
	Basketball_player	200	11	2925514	11	2925514	682
	Exercise	1	11	2391718	11	2391718	530
	Model	1	11	2458429	11	2458429	561
	Mitch	1	11	2289640	11	2289640	821
	ThomasSenic	170	11	2277443	11	2277443	749
	Football	1365600	11	1021107	11	1021107	160
	Façade 15	-	14	8907880	12	6834258	2517
	Façade 64	-	14	19702134	12	12755151	3539
	Egyptian_mask	-	12	272684	9	269739	141
	Head_00039	-	12	13903516	12	13903516	9218
	Shiva_00035	-	12	1009132	10	901081	282
	ULB_Unicorn	-	13	1995189	10	1588315	462
	Landscape_00014	-	14	71948094	12	15270319	3069
	Stanford_Area_2	-	16	47062002	12	19848824	4178
	Stanford_Area_4	-	16	43399204	12	24236048	6559
Validation	Boxer	viewdep	12	3493085	11	3056129	915
	Dancer	1	11	2592758	11	2592758	604
	Façade 09	-	12	1596085	11	1560834	836
	Frog_00067	-	12	3614251	11	3321097	1467

4.2 SR Model Training

The training of the DL 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 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 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 SR model is used for all the rates, unlike the DL coding model. As such, the only dependency is the sampling factor, with two DL 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 64×64×64, just as for the DL coding model; however, for sampling factor 4, the training PC dataset was divided into blocks of size 128×128×128 instead, resulting in a total of 9699 blocks for training and 1078 blocks for validation.

Implementation and training were done in PyTorch version 1.12. For training, the Adam algorithm [Kingma, 2015] was used with a learning rate of 10^{-4}, and minibatches of 2 and 16 blocks for sampling factor of 2 and 4, respectively.

5 Coding Configurations

In DL-based coding solutions, it is typical to train a DL 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, namely:

- **DL coding model:** Six DL coding models are available, which were trained for different RD trade-offs, spanning a wide range of rates.
- **Sampling factor (SF):** 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 (SR):** 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 (BS):** 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 (QS):** The quantization step parameter applied to the latents can be used for finetuning the target rate for each PC after selecting a specific DL coding model.

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.

6 References

[wg1/N100097] ISO/IEC JTC1/SC29/WG1 N100097, “Final Call for Proposals on JPEG Pleno Point Cloud Coding” Online Meeting, January 2022.

[wg1/M96005] A. F. R. Guarda, N. M. M. Rodrigues, M. Ruivo, L. Coelho, A. Seleem, F. Pereira, “IT/IST/IPLeiria Response to the Call for Proposals on JPEG Pleno Point Cloud Coding” ISO/IEC JTC1/SC29/WG1, Document M96005, Online Meeting, July 2022.

[Ballé, 2018] J. Ballé, D. Minnen, S. Singh, S. J. Hwang and N. Johnston, “Variational Image Compression with a Scale Hyperprior,” International Conference on Learning Representations (ICLR’2018), Vancouver, Canada, Apr. 2018.

[Szegedy, 2017] C. Szegedy, S. Ioffe, V. Vanhoucke and A. Alemi, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning,” AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, Feb. 2017.

[Minnen, 2018] D. Minnen, J. Ballé and G. Toderici, “Joint Autoregressive and Hierarchical Priors for Learned Image Compression,” Advances in Neural Inf. Process. Syst., Montreal, Canada, Dec. 2018.

[Akhtar, 2020] A. Akhtar, W. Gao, X. Zhang, L. Li, Z. Li and S. Liu, "Point Cloud Geometry Prediction Across Spatial Scale using Deep Learning," IEEE Int. Conf. on Visual Communications and Image Processing (VCIP), Hong Kong, China, Dec. 2020.

[Çiçek, 2016] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox and O. Ronneberger, "3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation," Int. Conf. on Medical Image Computing and Computer-Assisted Intervention (MICCAI), Athens, Greece, Jun. 2016.

[Quach, 2020] M. Quach, G. Valenzise and F. Dufaux, "Improved Deep Point Cloud Geometry Compression," IEEE Int. Workshop Multimedia Signal Process., Tampere, Finland, Sep. 2020.

[Lin, 2017] T. Lin, P. Goyal, R. Girshick, K. He and Piotr Dollár, "Focal Loss for Dense Object Detection," IEEE International Conference on Computer Vision (ICCV'2017), Venice, Italy, Oct. 2017.

[wg1/N100112] ISO/IEC JTC1/SC29/WG1 N100112, "JPEG Pleno Point Cloud Coding Common Training and Test Conditions v1.1", Online, January 2022.

[Bégaint, 2020] J. Bégaint, F. Racapé, S. Feltman, A. Pushparaja, "CompressAI: A PyTorch Library And Evaluation Platform For End-To-End Compression Research," arXiv:2011.03029 [cs.CV], Nov. 2020.

[Kingma, 2015] D. P. Kingma and J. Ba, "Adam: a Method for Stochastic Optimization," International Conference on Learning Representations (ICLR'2015), San Diego, CA, USA, May 2015.