Martins *et al.*
EURASIP Journal on Image and Video Processing (2024) 2024:15
https://doi.org/10.1186/s13640-024-00633-4

EURASIP Journal on Image and Video Processing

RESEARCH Open Access

Impact of LiDAR point cloud compression on 3D object detection evaluated on the KITTI dataset

Nuno A. B. Martins[1,2*], Luís A. da Silva Cruz[1,2] and Fernando Lopes[2,3]

*Correspondence:
nuno.martins@student.uc.pt

[1] Department of Electrical and Computer Engineering, University of Coimbra, Coimbra, Portugal
[2] Instituto de Telecomunicações, Coimbra, Portugal
[3] Polytechnic Institute of Coimbra, Coimbra Institute of Engineering, Coimbra, Portugal

Abstract

The rapid growth on the amount of generated 3D data, particularly in the form of Light Detection And Ranging (LiDAR) point clouds (PCs), poses very significant challenges in terms of data storage, transmission, and processing. Point cloud (PC) representation of 3D visual information has shown to be a very flexible format with many applications ranging from multimedia immersive communication to machine vision tasks in the robotics and autonomous driving domains. In this paper, we investigate the performance of four reference 3D object detection techniques, when the input PCs are compressed with varying levels of degradation. Compression is performed using two MPEG standard coders based on 2D projections and octree decomposition, as well as two coding methods based on Deep Learning (DL). For the DL coding methods, we used a Joint Photographic Experts Group (JPEG) reference PC coder, that we adapted to accept LiDAR PCs in both Cartesian and cylindrical coordinate systems. The detection performance of the four reference 3D object detection methods was evaluated using both pre-trained models and models specifically trained using degraded PCs reconstructed from compressed representations. It is shown that LiDAR PCs can be compressed down to 6 bits per point with no significant degradation on the object detection precision. Furthermore, employing specifically trained detection models improves the detection capabilities even at compression rates as low as 2 bits per point. These results show that LiDAR PCs can be coded to enable efficient storage and transmission, without significant object detection performance loss.

Keywords: Point clouds, LiDAR, Compression, 3D object detection, Autonomous driving

1 Introduction

Three dimensional point clouds (PCs) are a flexible format for the representation of 3D objects and scenes. Light Detection And Ranging (LiDAR) scanners can output data in PC format facilitating the processing when using the many machine vision algorithms developed in the last few years for applications in sensing for smart cities, robotics and automated driving [1]. However, modern LiDAR sensors generate very large amounts of data, with detrimental effects to the storage, transmission and processing of the captured PCs.

A possible solution to this problem is to introduce coding for compression in the LiDAR point cloud (PC) processing flow, as for example in autonomous vehicles, hopefully reducing the amount of data to manageable quantities. Two important requirements to be fulfilled by the LiDAR point cloud compression (PCC) technology are low impact on the performance of the processing operations done downstream and low computational complexity compatible with the use of resource constrained embedded computing platforms. This is not an easy task as LiDAR PCs acquired by rotating scanners are characterized by sets of points with spatially varying densities. The spatial density varies based on the complexity of objects or terrain, but also with the distance to the sensor, where PCs are denser near the sensor. Vertical structures, such as trees and buildings, are often represented by densely distributed points, contributing to a detailed understanding of the environment geometry. In terms of point organization, although the scanning patterns of LiDAR systems are typically fixed and well defined, irregular arrangements may occur in dynamic or complex environments. Additionally, LiDAR acquired PCs incorporate intensity or reflectance values, providing information about the surface properties of the scanned points.

There are several real-world important LiDAR PC data applications that would greatly benefit from efficient PCC. These include Terrain Mapping to represent terrain elevation profiles and geographical information, captured using for example an airborne LiDAR system, or Autonomous Navigation, where the problem of efficient PC processing and transmission is particularly relevant if inter-vehicle PC exchange or off-vehicle PC processing (e.g. on 5G edge computing facilities) are to be used [2].

The main objective of the study presented in this paper is to analyse the impact of state-of-the-art LiDAR PCC algorithms on the performance of reference object detection algorithms, fundamental for the detection of other vehicles and pedestrians in autonomous driving applications. Several studies addressing related problems have been published, as for example [3] where the impact of image compression on computer vision tasks is evaluated and, [4] where the effects of PCC on object detection using interpolated data from the cloud projection into 2D and PointNet++ [5], are compared.

In the present study, two standard and two DL-based PC coding algorithms were used to compress the KITTI 3D object detection dataset [6] at five compression rates. Four publicly available pre-trained 3D object detection algorithms were evaluated on the reconstructed/decompressed PCs and the detection results collected and analysed. As part of the study, the same 3D object detection algorithms were re-trained on the reconstructed/decompressed PCs, to understand if re-training using data with the same type of information loss due to compression as the test data, can improve the performance of the detectors. The reported results have special relevance for applications in use cases where LiDAR data need to be efficiently transferred and/or stored, to be further used for 3D object detection and other computer vision tasks. The inclusion of DL-based PC coding methods in our study, allows for an analysis on how the learned features and specific distortions associated with DL-based compression methods may affect DL-based object detection performance. In this context, the obtained results can be used to understand the tradeoff between data volume reduction brought by the compression and the degradation on the detection performance, helping LiDAR processing system designers choose the best compromise between those two variables.

The rest of the paper is organized as follows: Section 2 provides background information needed to understand PC coding techniques, including DL-based coding, as well as a description of 3D object detection methods. Section 3 an adaptation of the DL-based PC coding Joint Photographic Experts Group (JPEG) Verification Model needed to use LiDAR acquired PCs, in both Cartesian and cylindrical coordinates, including considerations for training dataset construction. In Section 4, the compression setup using the four coding methods is explained, along with a description of the used evaluation distortion metrics. This is followed by a discussion on some characteristics of the obtained reconstructed PCs and the achieved compression rates. The impact of the compression on 3D object detection is evaluated and discussed in Section 5, first using pre-trained models, and then using models that were re-trained using compression degraded PCs. Finally, Section 6 summarizes the obtained results, presents some conclusions and proposes directions for future research.

2 Background information

This section provides comprehensive background information, crucial to understand the subsequent sections, with a detailed overview of important PC coding methods and standards. It covers LiDAR specific methods as well as recent DL-based coding algorithms, including a solution proposed by the JPEG Pleno project. It also includes a review of state-of-the-art 3D object detection techniques.

2.1 Generic point cloud coding methods

Various approaches exist for PC data coding, which can generally be categorized into three main classes based on their coding principles. These include coding of 3D geometry information through direct 3D data processing, coding of 3D data via projection into 2D planes, and coding of 3D geometry information using graph representations. In this section, we primarily focus on the first two classes, which include methods employing 3D spatial decompositions such as octrees and techniques involving projections onto 2D spaces.

Often mentioned together, [7] and [8] propose very similar methods for static PCC using the octree data structure as the basis of their method, and employ predictive coding for surface approximation within a node. In [9], a method that encodes the first PC in a sequence and then only encodes the changes in point distribution for the next ones is presented. An improvement on the previously described compression scheme, by sorting the octree before performing the entropy coding task, is introduced in [10]. The advantage of organizing the nodes in the octree in ascending order is that the serialized output will have long sequences of repeated symbols, that typically yield high compression ratios. Temporal correlation between octree frames is also explored in [11], by considering the colour of the models as a graph signal. Correspondence is achieved by matching the spectral features of the nodes in consecutive octrees. A different partition method [12], explores the use of quadtree division of flat surfaces of the PC and octree division in non flat surfaces, the result being a hybrid tree with binary and quadtree nodes.

As for projection-based PCC, a real-time compression method is proposed in [13], using a pre-processing stage where the space is divided into user-defined size voxels,

that are converted individually into planar 2D domains, using height maps that are then compressed using JPEG. The authors of [14] and [15] present a scheme to convert PC surfaces into surface patches that are also parameterized as height maps. The main contribution of [16] is that a set of representative surface patches are chosen based on the similarity with others in the object surface. Only representative patches and their positions are encoded and transmitted. A two-part paper [17] and [18], describes a volumetric approach towards PCC, where a continuous volumetric B-spline function is defined as a surface by fitting it to the PC data. The function coefficients are then quantized and transmitted.

Following industry demand for efficient PCC, Moving Pictures Expert Group (MPEG) developed two PCC standards: video-based point cloud coding (V-PCC) and geometry-based point cloud coding (G-PCC) [19] based, respectively, on 2D projections and octree decompositions. V-PCC uses projections to flatten patches of the geometry into 2D patches and then encodes the sequence of 2D patches as video, making it adequate for use with dense PCs. In turn, G-PCC uses octree coding methods and data structures to encode the geometry of voxelized PCs.

2.2 Point cloud coding for LiDAR

LiDAR PCs, acquired by rotating scanners, exhibit varying spatial densities due to their specific structures and fixed scanning pattern. Typically, these PCs are denser near the sensor and sparser farther away. Such characteristics can be exploited for compression, with specific methods for this PC acquisition concept being proposed in the literature.

For instance, apart from the direct encoding and planar modes, G-PCC also has an angular mode specifically for LiDAR PCs. The angular mode in G-PCC has resulted from improvements to the planar mode when dealing with LiDAR acquired data, since it optimizes binary occupancy octree coding by utilizing sensor priors such as the position and number of lasers, as well as the angular resolution of each laser [20].

More recently, MPEG set out to develop a low complexity encoder specifically targeting LiDAR-acquired PC data. The so-called Low-Latency Low-Complexity LiDAR Codec (L3C2) was first proposed in [21] and exploits the acquisition order of LiDAR sensor priors in the horizontal and vertical directions. The horizontal direction corresponds to the azimuthal angle step for the acquisition of a laser. The vertical direction corresponds to the elevation angle step of each laser acquisition. The order of acquired laser samples fills a coarse 2D occupancy map, that is coded following a column-by-column lexicographic order, with a column representing the elevation angles for a given azimuthal angle step. The advantage of respecting the sensor acquisition order is that only the offset between two consecutive points has to be coded, to determine the position in the coarse representation.

In [22] and [23] LiDAR data are flattened into 2D range images that are then compressed using image compression methods. A Simultaneous Localization And Mapping (SLAM) approach towards predicting and compress consecutively acquired PCs is presented in [24]. In [25] the redundant raw packet data and structure produced directly by Velodyne LiDAR devices is exploited.

LASzip [26] is another compression tool targeted towards LiDAR PC data. It is a lossless and order-preserving codec that treats PCs as sequences mostly used in applications requiring large-scale LiDAR acquisitions.

2.3 Deep learning-based point cloud coding methods

Recently there has been much interest in the introduction of deep learning techniques into the compression field, following the pattern observed in many other areas. PC data compression is no exception to this trend, and there are several recently published proposals for data compression methods using learned models.

One of the early examples of DL-based PCC was the study presented in [27], where the authors made model optimizations based on a ratio and distortion parameter tradeoff, to create a static PC geometry compression technique based on learned convolutional transforms and uniform quantization. The same authors further improve their method in [28] in a way that, it trains the 3D convolutional neural network autoencoders with a learned prior. The improvements in PCC performance are achieved with the addition of features that include sequential training, the inclusion of focal loss, and a more efficient architectural implementation with the addition of residual blocks and deeper transforms, progressively increasing channels as the resolution decreases. In [29], the same authors propose an approach to compressing PC attributes. It treats the PC as a discrete 2D manifold in 3D space. The system utilizes 2D parameterization and grid mapping to leverage image processing and compression tools. This involves using a deep neural network as a parametric function to fold a 2D grid onto a 3D PC. Attributes from the original PC are then mapped to this grid, allowing for efficient recovery of PC attributes. Although the 3D-to-2D mapping creates some distortion, the authors describe strategies to mitigate this in practice.

In [30] a learning-based PCC method is presented making use of the variational autoencoder (VAE) concept with stacked Volumetric Rendering Networks (VRNs) and hyperpriors to improve coding efficiency. Also, during training, the method incorporates Weighted Binary Cross-Entropy (WBCE) loss. For inference, it uses adaptive thresholding to determine voxel occupancy. A second version of the method in [31] employs progressive down-scaling and sparse convolution techniques to enhance the efficiency of tensor processing. The downscaled representation effectively captures the sparse and unstructured characteristics of the points, while incorporating multiscale re-sampling to account for geometric structural variations. The geometry's latent representation is compressed using lossless octree compression methods. The proposed framework optimizes the bit rate and distortions at each scale, using the Binary Cross-Entropy (BCE) loss. With the usage of sparse tensors in [32] between representations of the same and different scales of a PC, the authors managed to greatly reduce the complexity when compared to the previous implementations. A cylindrical coordinate approach towards LiDAR PC representation based on [32] is made in [33], with latent features encoded using G-PCC, improving the original method for this type of representation.

Instead of using PC coordinate data, [34] uses raw packet data directly from a LiDAR sensor. The data are organized into a 2D image with the help of pitch and yaw information from the sensor, as described in [22]. After being normalized to 2D, data are fed to an autoencoder for spatial compression. The calculated residuals between the

reconstructed output from the decoder in the first iteration and the original input, are used for the subsequent iterations. The same authors further improve upon the previously described method by exploring the temporal redundancy of the streamed data [2]. As in typical video compression, the authors introduce the definition of I-frames (intra-predicted frames) and B-frames (bi-directionally predicted frames). The method is based on a U-net [35] that uses two I-frames to predict B-frames between them. The feedback to the network is given by the calculated residuals between predicted interpolation B-frames and real B-frames.

In [36], a point-based compression architecture that avoids discretization effects caused by using grid-based representations is proposed. It consists of an encoder that subsequently reduces the number of points and computes, for each point, a feature based on its local neighbourhood using kernel point convolutions (KPConvs) [37].

Following a call for proposals from JPEG [38], the first version of the so-called JPEG Pleno Point Cloud Coding codec (JPEG Pleno PCC) [39] was introduced. The JPEG Pleno PCC is a DL-based neural network compression method that has a unique joint geometry and colour coding system, that uses the same DL model to process both geometry and colour simultaneously. Given the relevance of this method to the work presented in this paper, a detailed description will be provided in Section 3. This will include an adaptation to encode LiDAR PCs in both Cartesian and cylindrical coordinates. Additionally, among other advancements, recent developments have introduced sparse convolutions to PC compression [40].

2.4 3D object detection methods

In this section, we review state-of-the-art 3D object detection methods that use LiDAR as their source.

3D Object detection methods using LiDAR PCs can be classified into four main categories: point-based, voxel-based, point-voxel, and range-based detection methods [41]. In the following, and in the scope of this classification, we present the main characteristics of four deep learning 3D PC detection methods that are considered state-of-the-art seminal works by introducing innovative concepts, incorporated into most of the best performing detection methods that directly use LiDAR data. This excludes range-based 3D object detectors from this study. These four methods are the ones selected in this work to evaluate the object detection performance when using compression degraded PCs, with results presented in Section 5.

An important class of 3D object detection methods process the PC raw points. These methods build upon the advances achieved by deep learning techniques like PointNet [42] and PointNet++ [5]. One example of a PointNet-based detection method is PointRCNN, that introduces a two-stage object detection framework combining region proposal generation and object detection proposals [43]. Another key feature of this implementation is the employment of a Region Proposal Network (RPN) that generates 3D proposals in the form of oriented 3D boxes. This RPN effectively narrows down the search space, allowing for better efficiency without sacrificing accuracy. More recent works such as [44] improve on this general framework to achieve higher performance, with some variations in the region proposal strategies.

In the case of voxel-based 3D object detection frameworks, input PCs are divided into equally spaced 3D voxels and the points within each voxel are transformed using either 2D or 3D convolutional neural networks. The resulting feature representation is then connected to a RPN to generate detections. VoxelNet [45] is a pioneering work that uses a voxel-based approach to 3D object detection. Based on similar ideas, the authors of SECOND (Sparsely Embedded Convolutional Detection) [46] use a sparse convolutional backbone network. To generate object proposals, SECOND first projects the 3D PCs onto multiple 2D bird's eye view maps, capturing the objects' spatial relationships from different perspectives, and then, a RPN is utilized to generate potential object locations based on the extracted features.

In PointPillars [47], the PC is divided into grids in the x-y coordinates, creating a set of vertical pillars [48] to produce a 2D bird's eye view of the scene. Additionally, PointPillars training uses data augmentation techniques specifically designed for PC data. These techniques include random flipping, scaling, and rotation, which effectively increase the diversity of the training data and enhance the model's generalization capabilities. The partition of the PC into pillars is also used in more recent works such as [49] and [50].

PointVoxel-RCNN (PV-RCNN) [43] follows a combined approach that leverages the advantages of efficient generation of detection proposals, that are associated with voxel-based 3D object detection, and more rich PC features associated with point-based 3D object detection. PV-RCNN also introduces a dynamic voxelization strategy that adaptively adjusts the voxel size based on the object scale, ensuring accurate representation of objects with varying sizes. This adaptability contributes to improved detection performance for objects at different scales. PV-RCNN++ [51] further improves on the method by changing the regions of interest generation strategy, and introduces *Vector-Pool Aggregation* for better aggregation of local point features.

3 Modified JPEG Pleno PCC coder

We use the JPEG Pleno PCC coder as a representative DL-based approach to compress and reconstruct PCs for object detection performance evaluation. However, in its original form, it has the inability to process the reflectance attribute inherent to LiDAR data. Given that the object of this work is to evaluate the impact of the compression on 3D object detection methods that rely on this attribute for object detection, an adaptation was essential to ensure compatibility and a fair comparison. In this section, we present the main components and the modifications made to the DL-based JPEG Pleno Point Cloud Coding Verification Model (VM) V1.0 [39]. The level of detail in this section is compatible with the description of the modifications to handle the reflectance, and with the need for a clear definition of the training procedures.

The encoder follows a sequential process according to the general architecture in Fig. 1. Initially, the module "PC Block Partitioning" divides the input PC geometry into 3D blocks that are treated as independently coded units with a designated block size (BS). Subsequently, in the "Block Down-Sampling Module", these blocks may undergo optional down-sampling to a lower grid precision, using a specified sampling factor (SF) to increase block density before coding. This down-sampling serves as a tool to achieve lower rates and can yield compression gains, particularly for sparse PCs. The final encoding step in "DL-based Block Encoding" involves coding each 3D block through a

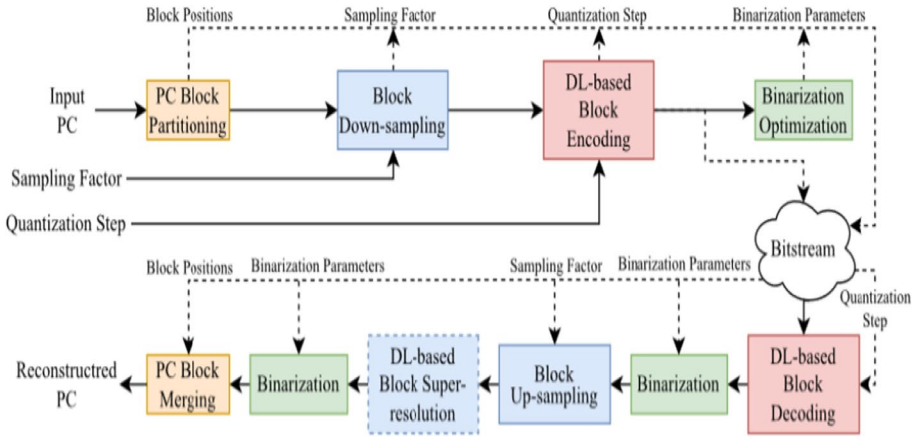


Fig. 1 Original architecture of the JPEG Pleno PCC (from [39])

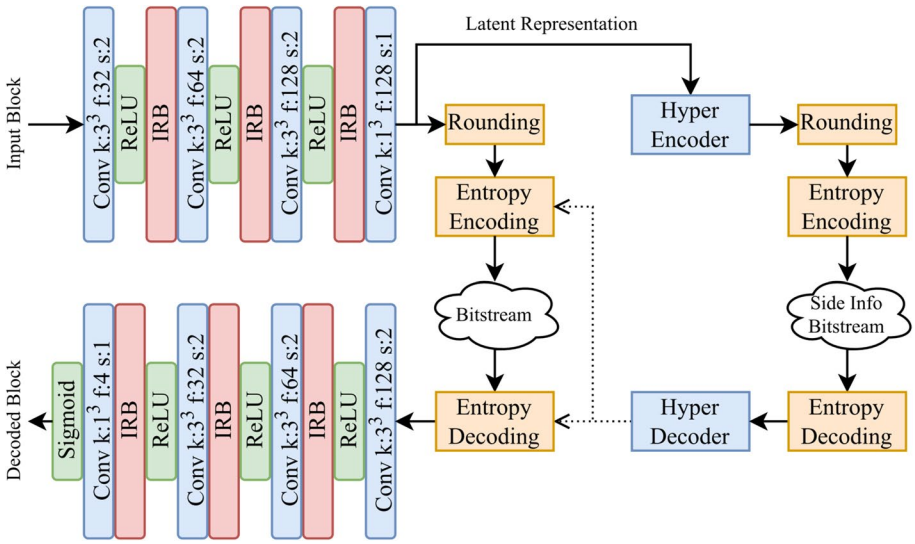


Fig. 2 "DL-based Block Encoding" model description (from [39])

DL model according to the architecture illustrated in Fig. 2, where a non-linear transform with multiple 3D convolutional layers progressively reduces data dimensionality until reaching the bottleneck layer. Additionally, Inception-Resnet Blocks (IRBs) [52] are incorporated to extract features by employing parallel convolutional layers with varying filter support sizes. Following quantization of the feature rich latent representation at the autoencoder bottleneck, adaptive entropy coding is executed through a secondary network, tasked with extracting information from the latent representation and generating a more precise entropy model.

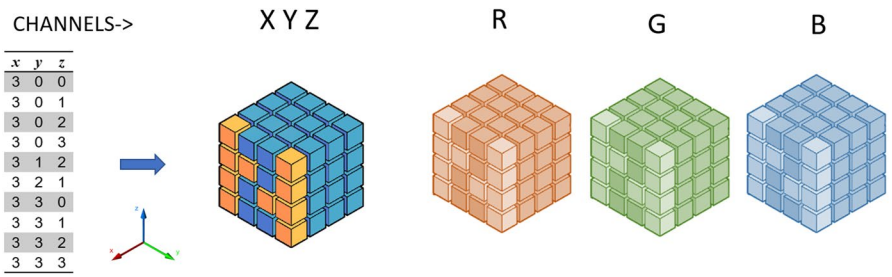
On the decoder side, starting with the "DL-based Block Decoder" the representation of the encoded blocks is transformed back to a degraded representation of the original input PC blocks, using the trained DL model. The "Basic Block Up-Sampling" module applies the inverse up-sampling to convert the blocks back to the original PC precision. The "DL-based Block Super Resolution" module, not utilized in the

analysis presented in this paper, serves as an optional post-processing step, designed to densify PCs already processed by the Block Up-sampling module, thus enhancing the reconstructed quality without increasing the compression rate. Finally, the reconstructed blocks are merged back together to form the representation of a completely reconstructed PC in the "PC Block Merging" module.

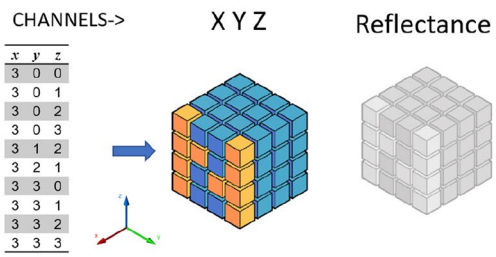
An important notion to have about this technique is how the PC input information is handled and represented within the method. Before encoding, a PC is transformed into a 3D block representation, which is based on voxels to form a regular structure. The input PC has data which include geometry and colour components in RGB colour space. So, every voxel features four channels, of which the first is a binary signal illustrating the geometry information; a '1' stands for an occupied voxel and a '0' for an empty one. The next three channels are represented in a similar fashion, where occupied voxels are given the corresponding value of each one of the RGB channels scaled down from the range [0, 255] to [0, 1]. So, a PC with attributes can then be divided into separate blocks which can then be coded separately using the DL coding model, as illustrated in Fig. 3a.

For the adaptation of the method to accept LiDAR reflectance attributes the same voxel occupation representation for geometry information is kept, and reflectance for occupied voxels is represented as a single attribute channel also scaled down from the initial range [0, 255] to [0, 1] as shown in Fig. 3b. Encoding is done for each one of the individual partitioned blocks of the original PC.

To maximize the compression performance, the model is trained by minimizing a loss function that takes into account the distortion of the decoded blocks in relation to the input blocks, as well as, the estimated coding rate. For this, the loss function of the original proposal [39] is maintained:



(a) JPEG Pleno PCC original representation of voxel occupation (from [39]).



(b) Adaptation of JPEG Pleno PCC made to process LiDAR reflectance attribute.

Fig. 3 Representation of voxel occupation for geometry + colour in the original method (**a**), and the adaptation made to process the single LiDAR reflectance attribute (**b**)

$$LossFunction = Distortion + \lambda.CodingRate, \tag{1}$$

where λ is a Lagrangian multiplier that sets a target Rate-Distortion(RD) point for each one of the DL coding models to be trained. This means that models using different values of λ have to be trained for every required compression ratio. The total distortion is given by:

$$\text{Total Distortion} = (1-\omega) \times Distortion_{Geometry} + \omega \times Distortion_{Reflectance}, \tag{2}$$

where ω sets the weight given to the reflectance over the geometry. For this study $\omega = 0.1$ has been used, following initial experiments that showed that giving more importance to the $Distortion_{Geometry}$ component of the *TotalDistortion* function, produced better compression with low impact in the reflectance objective quality of the decoded PCs. The method uses the Focal Loss (FL) to determine geometry distortion as follows:

$$FL(v,u) = \begin{cases} -\alpha(1-v)^{\gamma} log(v), & u = 1 \\ -(1-\alpha)v^{\gamma} log(1-v), & u = 0 \end{cases} \tag{3}$$

where u is the original voxel value and v is the probability value of the corresponding voxel value. α is a weight to control class imbalance and γ defines the importance of correction of misclassified voxels in relation to correctly classified voxels. The proposed values of $\alpha = 0.7$ and $\gamma = 2$ were left unchanged throughout the entire set of experiments.

In turn, the distortion function for the reflectance ($Distortion_{Reflectance}$) was introduced to the method, and is a voxel-wise mean squared error as follows:

$$Distortion_{Reflectance} = \frac{1}{N_{input}} \sum_{i \in N_{input}} \left(Reflectance_i - Reflectance_i^{'} \right)^2, \tag{4}$$

where N_{input} is the number of occupied voxels in the input block, $Reflectance_i$ is the reflectance value of the occupied voxel i in the input block and, $Reflectance_i^{'}$ is the inferred reflectance value of the corresponding voxel in the decoded block.

3.1 Training procedure for Cartesian coordinates

As the intent is to encode LiDAR PCs, the modified JPEG Pleno PCC coder must be trained to use learned features that are particular to this sparse PC representation. For that, the KITTI 3D Object Detection dataset was used to build a training dataset. The KITTI dataset is widely known in the field of autonomous driving and computer vision for benchmarking 3D object detection algorithms. It contains an extensive set of real-world scenarios captured on vehicle-mounted sensors, including images and LiDAR PCs, along with a comprehensive annotation of object instances within the captured scenes, including the 3D bounding box coordinates and their corresponding class labels. In terms of PCs, the KITTI dataset contains a total of 7,481 training and 7,518 testing samples.

Originally, PCs in the dataset contain 3D coordinates (x, y, z) and *reflectance*, each value expressed in a 4-byte floating-point number, meaning an original processing representation with 128 bits per point. For the coding experiments, a 12-bit integer voxel grid was used to voxelize the entire dataset. While many works in the literature

commonly use 18-bit voxelization to ensure the preservation of geometric information within LiDAR PCs, an exploratory study done by the authors showed that adopting a 12-bit voxelization preserved almost all the points in the original non-voxelized PC. In fact, voxelizing the PCs in the KITTI dataset to 12-bit depths retained about 96% of the points in the original PCs, as reported in Table 1. As for the *reflectance* attribute, an 8-bit normalization was used, with values of original points contained within the same voxel averaged. Apart from maintaining data integrity, this choice positively impacts execution performance for compression methods that use block partitioning and encoding. The adoption of 18-bit voxelization would result in an increase in the number of blocks to encode with a low number of points each, rendering execution within a reasonable timeframe impractical.

To train the model on LiDAR data, the voxelized PCs must first be characterized. For KITTI 3D object detection, each PC was partitioned into 64x64x64 (*xyz*) blocks, resulting in a total of 25,333,760 blocks. From Table 2, on average, there are approximately 33.88 points per block and the maximum number of points per block is 8,625 and, from Table 3, 95% of all blocks have at most 150 points per block. Subsequently, to form the training dataset, we established one class for each point count per block, ranging from 1 to 150. To form each class, we populated 200 blocks with the initial 200 block

Table 1 KITTI 3D object detection voxelization vs. average preserved points per PC

Voxelization grid (bits)	Preserved points	%
18	119,224.68	100
16	119,224.68	100
14	119,218.58	99.99
12	114,736.50	96.24
10	68,649.24	57.58

Table 2 KITTI 3D object detection training dataset description in terms of partition blocks of size 64

Coordinate space	Cartesian	Cylindrical
Number of blocks	25,333,760	19,318,270
Average points per block	33.881	44.443
Standard deviation	104.303	81.806
Minimum number of points per block	1	1
Maximum number of points per block	8625	1726

Table 3 KITTI 3D object detection training dataset point distribution within partition blocks of size 64

Coordinate space	Cartesian	Cylindrical
Percentile (number of points per percentage of total blocks)		
25%	2	3
50%	6	13
75%	21	47
90%	70	127
95%	150	202

occurrences in the KITTI training dataset, for the corresponding number of points. For instance, class 1 comprises the first 200 64x64x64 blocks with a single point, while class 2 includes the initial 200 blocks with two points, and so forth. Furthermore, 10% of the blocks were set aside exclusively for model validation purposes during the training phase. This approach created a total of 27,000 training blocks and 3,000 validation blocks, representing approximately 0.11% of the overall number of blocks in the dataset.

3.2 Training procedure for cylindrical coordinates

For the case of LiDAR PC acquisition, the sensor uses a horizontal rotational movement in fixed increments and scans consistently across the vertical plane, resulting in the generation of data points. In an autonomous driving situation, the nature of objects often causes them to appear at the same horizontal level. Specifically, these PCs tend to exhibit denser points in close proximity to the sensor, gradually becoming sparser as the distance increases. Taking advantage of the fact that a cylindrical geometry is closer to the natural characteristics of the LiDAR sensor acquisition, the cylindrical coordinate system is a natural fit to represent this type of data, since all points exist within a circular acquisition range. In [33], the authors use the cylindrical coordinate system on a DL-based coding method improving on a Cartesian approach. Given the above reasoning, we also included in this work the training of the modified JPEG Pleno PCC using cylindrical coordinates.

Cylindrical coordinates are a three-dimensional coordinate system that represents a point in space using three parameters: radial distance (ρ), azimuthal angle (θ), and height (z). The radial distance ρ, is the distance from the origin to the point of interest, measured along a line that is perpendicular to the z-axis. Azimuthal angle θ is the angle measured in the xy-plane representing the rotation around the z-axis from a reference direction. The height z represents the vertical position of the point, measured along the z-axis from the xy-plane. Another advantage of using this representation is that, in practice, the acquired points would not need to be converted to a Cartesian xyz coordinate system before being used for downstream processing tasks.

Since all available datasets use the Cartesian coordinate system, one must first convert the PCs into cylindrical coordinates using the formula:

$$\rho = \sqrt{x^2 + y^2}, \theta = \arctan\left(\frac{y}{x}\right), and\ z = z. \tag{5}$$

Figure 4 shows the result of the conversion from the Cartesian coordinate system to the cylindrical coordinate system.

Following the training procedure employed for the Cartesian coordinate case, the training dataset from the KITTI 3D object Detection benchmark is firstly converted to the cylindrical coordinate system and also voxelized to the integer grid bit depth of 12 bits, as in the Cartesian coordinate case, resulting in a total of 19,318,270 blocks with a size of 64x64x64 ($\rho\theta z$). For this case, on average, each block contains 44.44 points. Expectedly, from Table 3, 95% of these blocks have at least 200 points, while the maximum number of points in any given block is 1,726. This time, to cover 95% of the block configurations present in this partition configuration, the training dataset contains, in this case, 200 classes with each class having 200 examples of block occupation

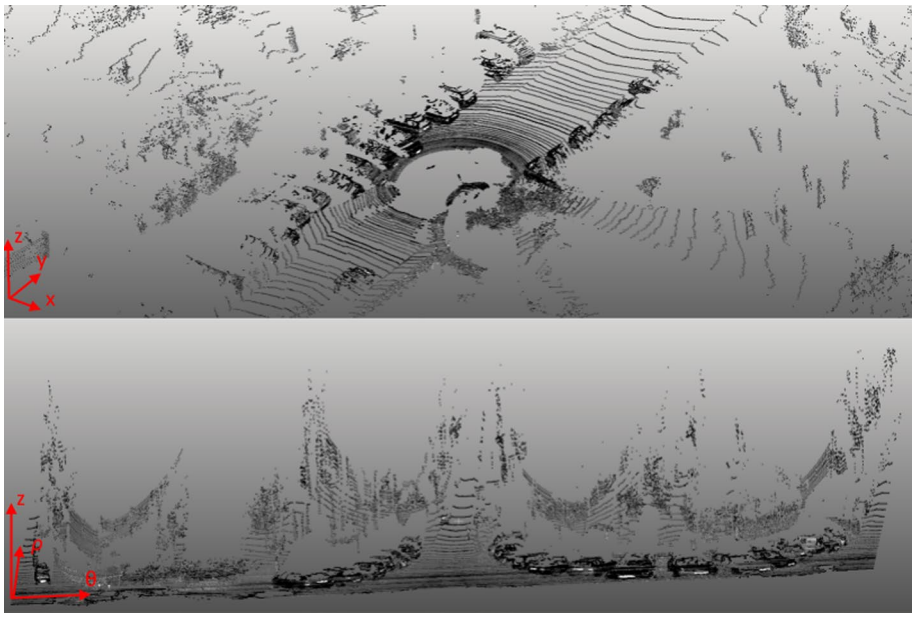


Fig. 4 Illustration of the conversion of a PC from Cartesian (top) to cylindrical coordinate system (bottom) where azimuthal angle (θ) is mapped to the x axis, radial distance (ρ) is mapped to the y axis, and height (z) is mapped to z axis

distributions. Again, 10% of the blocks were used for validation purposes. As a result of this conversion and representation in cylindrical coordinates, we can observe that this scheme produces a denser representation of the scene, as the number of total partitioned blocks is smaller while the number of points per block increased, when compared to the same information represented in the Cartesian coordinate system.

The networks were trained from scratch using the corresponding datasets. λ values of 0.000025, 0.00005, 0.0003, 0.0001 and 0.0006 were used for training. The sequential procedure of re-training subsequent compression rate models from the smallest to the largest λ, was used as described in the original implementation.

4 Experimental setup, coding configurations and test conditions

In this section, we present the experimental setup, PC coding configurations and test conditions that will be used to prepare the PCs and evaluate the impact of the compression on the computer vision task under consideration. The selected coding methods are the previously described modified JPEG Pleno PCC coder, both for Cartesian (MJ L-PCC) and for cylindrical coordinates (MJ LC-PCC), G-PCC version 14 [53] in angular mode, as well as L3C2. For each of the selected compression methods, five different compression parameters were selected to generate compressed datasets with varying levels of data reduction and associated objective quality.

Compression parameters for geometry and attributes for both G-PCC and L3C2 encoders, are shown in Table 4. Attributes were encoded using the *Hierarchical Neighbourhood Prediction as Lifting Transform* mode for both encoders, and G-PCC was configured to use the angular mode for geometry encoding. Other important parameters for the encoders are the priors related to the position of the LiDAR sensor head, as well as the vertical angle for each laser beam and the number of acquisitions per head turn.

Table 4 G-PCC and L3C2 encoder settings for the used compression ratios

	R1	R2	R3	R4	R5
Position quantization scale	$\frac{1}{1024}$	$\frac{1}{704}$	$\frac{1}{576}$	$\frac{1}{448}$	$\frac{1}{320}$
Attributes quantization parameter	51	43	37	32	30

These priors were parameterized according to the KITTI sensor calibration file since the laser positions differ from the default configurations provided with the MPEG CTC [54].

JPEG Pleno PCC for LiDAR used the previously trained models with λ values of 0.000025, 0.00005, 0.0003, 0.0001 and 0.0006, and a SF of 1 for PCs represented in both Cartesian and cylindrical coordinates, and a for a BS of 64.

The results of these experiments in terms of PC reconstruction distortion are presented in Section 4.2.

4.1 Distortion metrics

To evaluate the quality of the coded PCs, the recommended objective quality metrics PSNR D1, PSNR D2 and reflectance PSNR are used [54], as well as the density-to-density distortion—PSNR D3 [55].

When using point-to-point metrics, the distance between each point in the reference PC and the point that is closest to it in the reconstructed PC is computed. The most common distance measure used is the Euclidean distance, which calculates the straight-line distance between two points in a three-dimensional space. It is important to note that the point-to-point metric solely focuses on the position of individual points and does not take into account other factors such as surface normals or semantic information. As a result, it may not capture all aspects of quality or similarity in complex 3D scenes.

To complement the D1 quality metric, the D2 point-to-plane metric, that is based on the distance between a point and the corresponding plane, is used. The plane is determined based on the neighbouring points around each point in the reference PC. Then, for each point in the degraded PC, the distance between the point and the corresponding plane is computed. The point-to-plane metric provides a more comprehensive evaluation of the alignment quality because it takes into account not only the position of individual points, but also the local surface geometry. For PSNR D1 and D2 distortion values, the PSNR is defined as the peak signal over the symmetric distortion, computed as [54]:

$$PSNR_{Dx} = 10\log_{10}\left(\frac{3p^2}{max(e^{Dx}_{B,A}), max(e^{Dx}_{A,B})}\right), \tag{6}$$

where $p = 2^{bitdepth} - 1$ is the associated resolution of the PC voxelization, in this case 12 bits are used. $e^{Dx}_{B,A}$, $e^{Dx}_{A,B}$ are the mean squared errors depending on the PC being used as the reference—PC A being the original and PC B being the reconstructed PC—computed according to the D1 metric (point-to-point) or D2 metric (point-to-plane).

For the reflectance attribute quality metric, the symmetric computation of the reflectance mean square error (MSE) (*symmetricMSE*) is done following the same approach used for geometric distortions. The highest level of distortion of the two passes is then chosen as the measure of distortion:

$$PSNR_{Reflectance} = 10 \log_{10} \left(\frac{p^2}{symmetricMSE} \right), \tag{7}$$

where the peak value p in this case is 255 as the reflectance component of all used PCs has a bit depth of 8 bits per point.

The D1 point-to-point quality metric, the D2 point-to-plane metric and the reflectance PSNR values were calculated using the evaluation metric software for PC coding [56]. It is important to note that this tool assumes a 16-bit peak quantization value for the reflectance attribute. In our case, the tool was adapted to accurately determine the 8-bit reflectance PSNR.

The PSNR D3 measure described in [55] is a density-to-density measure that is based on the comparison of the input and reconstructed PC density distributions. It was specifically designed to detect density distribution degradation such as incorrect occupancy estimation. These occurrences are most common in DL-based coding methods such as the evaluated LiDAR JPEG Pleno PCC adaptation.

Finally, the number of bits per point from the input PC (*bpp*) is used as reference to measure the compression ratio.

4.2 Discussion on compression performance

This section presents an analysis of the rate-distortion performance of the PCs codings using G-PCC, L3C2 and both JPEG PCC LiDAR adaptations, tested according to the description at the beginning of Section 4. It describes the tradeoff between compression efficiency and reconstruction quality for each coding method. Figure 5 shows the results for the D1 point-to-point metric, D2 point-to-plane metric, D3 density-to-density metric and the reflectance attribute PSNR. Each point in the plots corresponds to the average PSNR for a given quality metric and achieved compression bit rate for every PC of the KITTI dataset. The error bars represent the standard deviation of the data points for both quality and bit rate. We included these error bars to account for the variability in quality and compression performance observed across different PCs within the dataset, which is influenced by the specific features present in each acquired scene.

Analysing the rate distortion (RD) performances in Fig. 5—point-to-point D1 and point-to-plane D2, it is clear that on average G-PCC and L3C2 outperform the DL methods in terms of geometry coding distortion performance. The same can be said for the reflectance attribute coding distortion according to Fig. 5—reflectance PSNR, while both JPEG PCC LiDAR adaptations, MJ L-PCC and MJ LC-PCC average PSNR values indicate good performance for the implemented methods. Also, Fig. 5—density metric D3 indicates that MJ L-PCC and MJ LC-PCC can produce reconstructed PCs with a density distribution pattern similar to that of the input. However, with an increase in both the local and global number of points (Fig. 6).

Comparing both non-DL methods, it is clear that the RD performance of G-PCC in angular mode, on average, is better than L3C2. But it can be said that the performance of

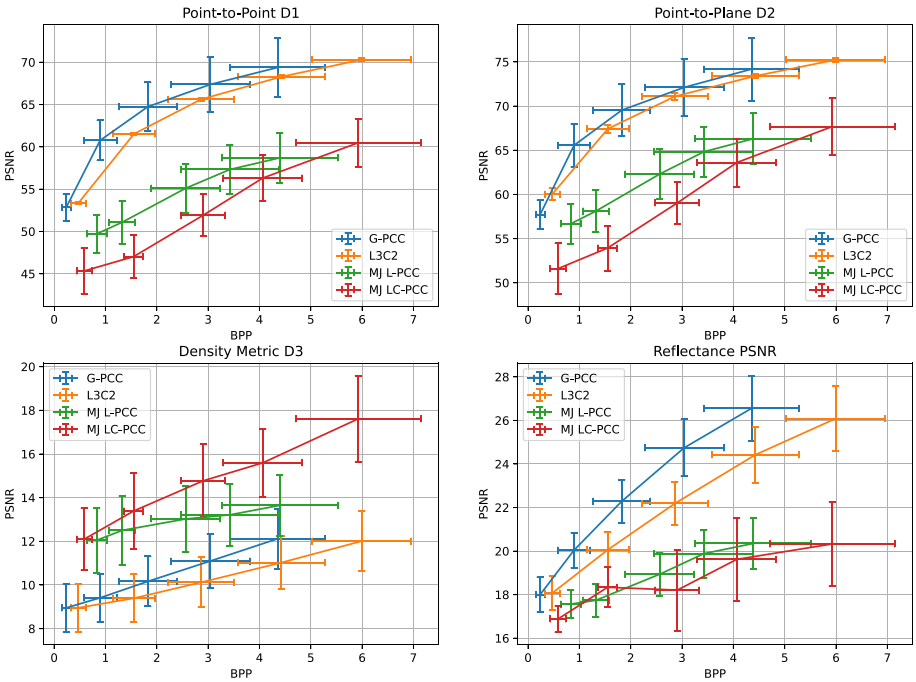


Fig. 5 Point-to-point D1 PSNR, point-to-plane D2 PSNR, density-to-density D3 PSNR, and reflectance PSNR for the KITTI 3D object detection benchmark

Fig. 6 Scene "003237" from the training set of KITTI 3D object detection benchmark, with examples of "Car", "Pedestrian" and "Cyclist" classes. To be used as visual reference to PCs shown in Figs. 7 and 13

G-PCC is highly dependent on the scene being compressed as for any given compression ratio there is a considerable deviation from the average value. This is due to the way the angular mode works, as it tries to approximate all points to a set of planes defined by the sensor head laser angle priors. Figure 7c, d shows this behaviour with the reconstructed PC having points placed in a well-ordered grid, that preserves the structure of objects but eliminates fine contour details. This gives G-PCC an advantage when encoding scenes containing regular surfaces as buildings, but poor performance in complex scenes in rural or forest areas. On the other hand, L3C2 can produce accurate reconstructed PC representations relatively independent of the morphology of the scene for a given target compression ratio. This is indicated by the error bars in Fig. 5—point-to-point D1 and point-to-plane D2 that present a small deviation from the average RD values for L3C2. This effect can be seen in Fig. 7e, f.

For the case of MJ L-PCC and MJ LC-PCC, they both perform worse than G-PCC and L3C2 in RD geometry metrics D1 and D2. This can be attributed to the fact that this DL

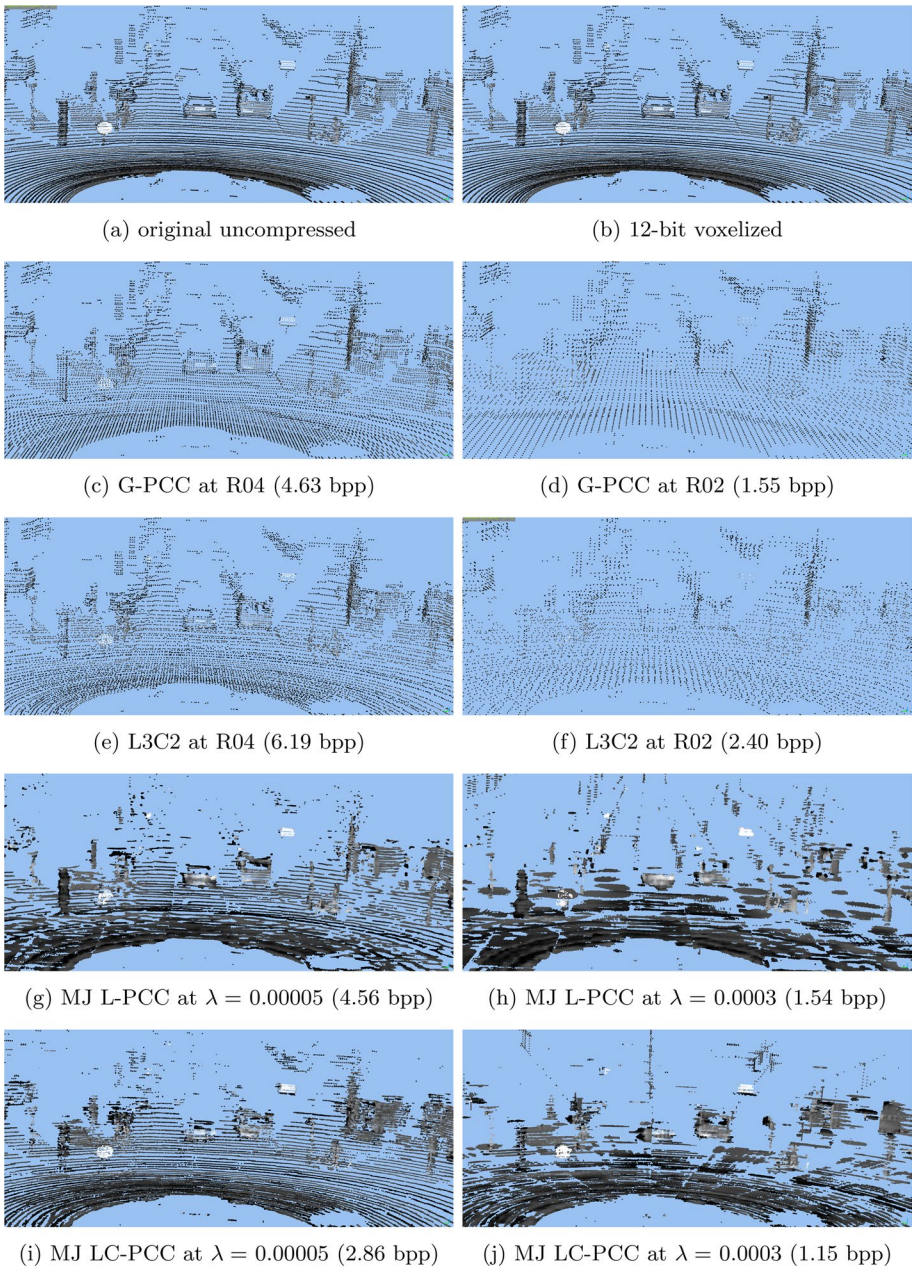


Fig. 7 Detail from PC "003237" from the training set of KITTI 3D object detection benchmark, compressed with G-PCC, L3C2, MJ L-PCC and MJ LC-PCC. "Low compression" refers to $\lambda = 0.00005$ and "high compression" refers to $\lambda = 0.0003$

method uses a "top-k" approach based on the voxel occupancy probability to determine whether a point should be added to the reconstructed decoded representation or not. This adds points to the vicinity of where the original point actually was, increasing the density in that area. This is also shown by the D3 metric, as the PSNR values indicate that MJ L-PCC and MJ LC-PCC can accurately replicate the original PC density distribution, where denser areas of the original PC are also denser areas in the reconstructed PC. Fig. 7h and j shows a detail of this behaviour. This increased density may be helpful

in the case of dense PCs where the presence of a point generally means that a surface is present, but less in the case of sparse LiDAR PCs where this is not true. This behaviour can be confirmed by looking at the reconstruction where regular structures such as roads are maintained, while intricate details disappear. Another obvious shortcoming is the ability of the method to represent points along the partition block boundaries, with the introduction of artefacts either by adding points to the boundary in sparser blocks or not representing them in denser ones. The consistency of the output is another problem. For instance, Fig. 7 features two cars exhibiting similar structures at a close distance, so one would assume a reconstruction with comparable characteristics for both structures as it is the case for G-PCC and L3C2. Yet this does not happen for MJ L-PCC and MJ LC-PCC. The division of the PC into blocks also aggravates the problem with the potential for an object of interest to be divided across contiguous blocks. This introduces degradation along the partition boundary affecting the representation of the object.

In turn, MJ LC-PCC performs worse than MJ L-PCC in terms of RD, even though intuitively the denser cylindrical representation should benefit the ability of the method to reconstruct PCs more faithfully to the original. This discrepancy in RD can be explained as the results shown are computed after converting the reconstructed PCs from cylindrical to Cartesian coordinate space. Consequently, even though the method works to reduce the error in cylindrical coordinate space, the conversion to Cartesian coordinate space does not correlate well with the geometry characteristics of the original input PC. In the particular case of MJ LC-PCC, although artefacts exist also in the block partition boundaries, they are distributed in the azimuthal axis of the LiDAR and widens with increasing distance from the sensor. Thus, there is a greater likelihood of an element of interest being entirely contained within a single block.

5 Impact of compression on 3D object detection

Following the reconstruction of the PCs using G-PCC in angular mode, L3C2, MJ L-PCC and MJ LC-PCC, this section describes the procedure to determine the impact that the use of degraded PCs has on 3D object detection method performance.

As described in Section 3.1, a 12-bit voxelized version of the KITTI 3D object detection training set is used to determine the performance of the detection methods on the reconstructed PCs subjected to the compression methods. It features annotated ground truth bounding boxes for each object class—"Car", "Cyclist" and "Pedestrian"—and each one is categorized in "Easy" (minimum object size of 40 Px, fully visible and max truncation of 15%), "Moderate" (minimum object size of 25 Px, partly occluded and max truncation of 30%) and "Hard" (minimum object size of 25 Px, difficult to see and max truncation of 50%). The official evaluation detection metric for this dataset is the 3D average precision (3D AP) and relies on the intersection over union (IoU) between predicted bounding boxes and ground truth bounding boxes. Only detections with a bounding box overlap of at least 70% for cars, and 50% for cyclists and pedestrians are considered for performance estimation on detection. Also, the result of the "Moderate" evaluation is used to rank a detection method submitted to evaluation on the benchmark. Following the standard literature practice, the official training dataset was divided into 3712 training samples and 3769 validation samples [57].

The four 3D object detection methods, SECOND, PointPillars, PointRCNN and PV-RCNN, were chosen, given their seminal work status in the literature, and each representing different approaches to the detection problem, that could be impacted differently depending on the compression method and compression parameters used: compression rate for G-PCC and L3C2; λ for MJ L-PCC and MJ LC-PCC.

The compressed PCs for each of the five compression rates and for the four encoding methods, G-PCC, L3C2, MJ L-PCC and MJ LC-PCC, were reconstructed and converted back to the original coordinate system and format of the KITTI dataset. Then, all four 3D object detection methods were applied to the reconstructed and converted PCs, and their detection performance was evaluated using publicly available pre-trained models [58]. With this experiment, we evaluated if learned PC features that are used by the pre-trained detection models are lost to some extent during the compression, affecting object detection performance. Figure 8 summarizes this experiment.

Another experiment was conducted, to evaluate to which extent the detection models' performance when using compression degraded PCs, could improve by being re-trained on reconstructed PCs after compression. With this approach, the re-trained models could learn new PC features introduced by the compression using PCs reconstructed on a particular compression parameter. To this end, all detection methods were re-trained using each individual reconstructed dataset (divided by compression parameters) from G-PCC, L3C2, MJ L-PCC and MJ LC-PCC, as the source. The pre-trained models were used as a starting point and re-trained for 140 epochs. The best performing epoch of the re-trained detection models were then evaluated using the corresponding reconstructed validation set from KITTI, as shown in Figure 9.

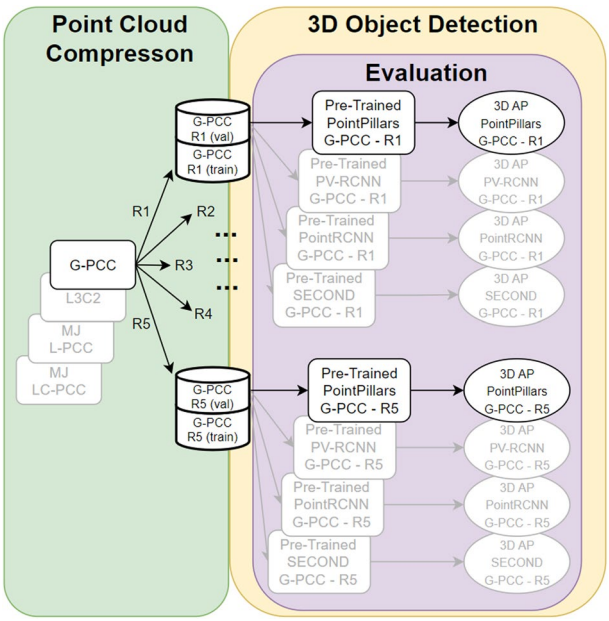


Fig. 8 Flowchart for the procedure to evaluate the pre-trained object detection performance on compressed PCs

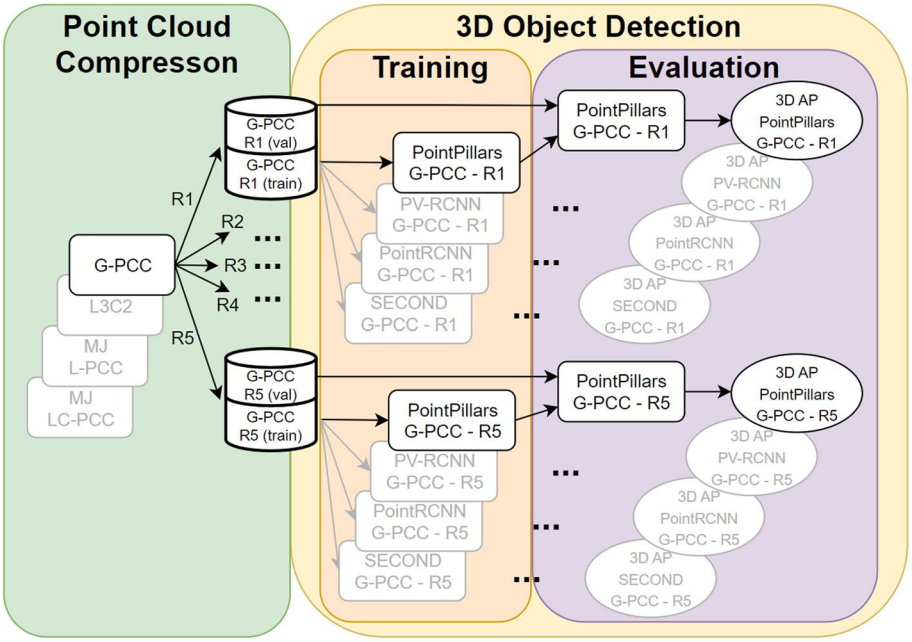


Fig. 9 Flowchart for the re-training and evaluation procedure to determine object detection performance on compression degraded PCs

5.1 Compression effects on 3D object detection using pre-trained models

The results of evaluating 3D object detection methods using pre-trained models are shown in Fig. 10. It can be observed that the performance for all tested detection techniques, difficulty levels and for both G-PCC and L3C2, suffer only a small degradation, even for very high compression ratios, down to about the average coding rate of 3 bpp for the "Car" class, 4 bpp for the "Cyclist" class and 5 bpp for the "Pedestrian" class.

Exceptions are PV-RCCN with the "Pedestrian" class where detection degradation is observed right from 6 bpp down, and PointPillars, also with the "Pedestrian" class, which has poor detection performance for the entire rate range. On the other hand, for the case of "Car" and the "Easy" category, degradation in detection starts to be noticed only near the 3 bpp rate mark.

Interestingly enough, PointRCNN detection performance on compressed PCs with G-PCC or L3C2 is slightly higher than that obtained on original uncompressed PCs. This is probably due to a filtering-like effect of the compression, that eliminates outlier/noisy points.

On the comparison between 3D object detection performance using G-PCC or L3C2, for the same rates, PCs encoded with G-PCC lead to better object detection performance than those encoded with L3C2. This should be expected since, according to Fig. 5—point-to-point D1 and point-to-plane D2, G-PCC produces, on average, better RD results for any given compression ratio, but looking into the error bars, L3C2 produces a reconstruction with less variation in objective quality independently of the input PC.

The better performance of G-PCC means that the detection methods do not use fine details to define the structure of an object of interest, so the features of G-PCC in

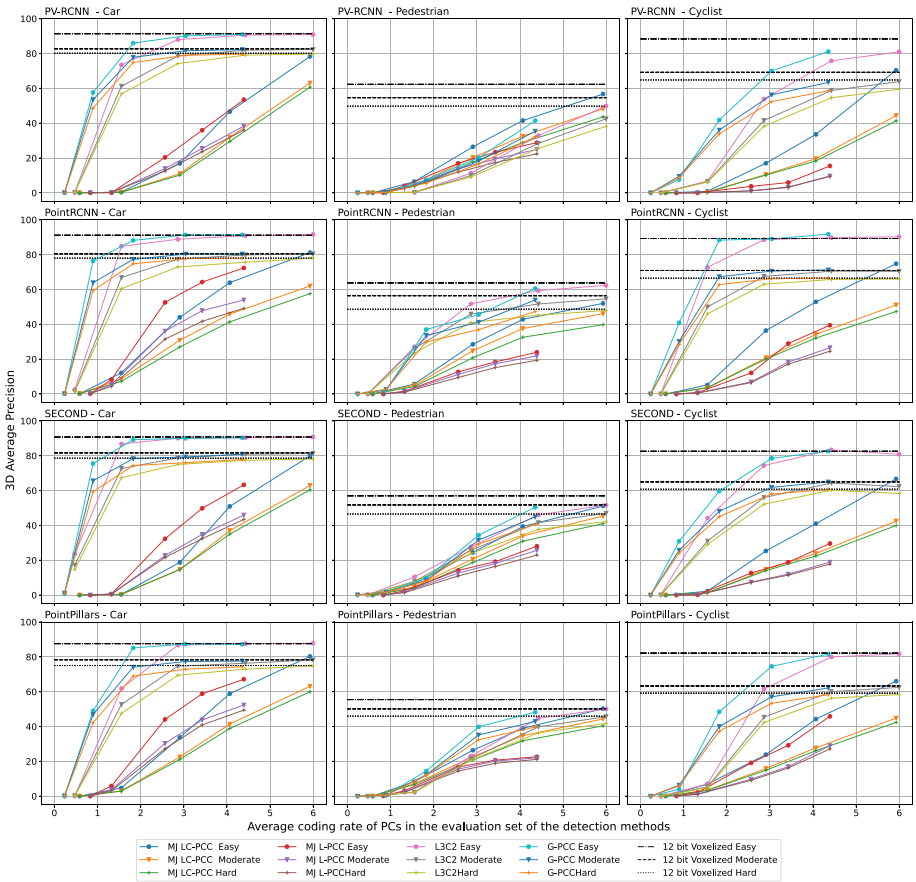


Fig. 10 3D average precision results for each object class and detection method evaluated on pre-trained models—method and class on top of each plot

angular mode produce a compatible PC representation that preserves relevant object features resulting in a more successful detection.

For the case of the DL-based coding methods, MJ L-PCC and MJ LC-PCC, the pre-trained 3D object detection methods struggle for all tested λ values. It is only for the case of "Pedestrian" PV-RCCN and SECOND, that these methods near the detection performance for G-PCC and L3C2. Comparing both DL coding methods, MJ L-PCC outperforms MJ LC-PCC for the "Car" class, while for the other two classes, MJ LC-PCC has the best performance.

MJ LC-PCC even has the best performance on PV-RCNN "Pedestrian", and achieves similar results to G-PCC and L3C2 for SECOND and PointPillars. The DL-based compression methods produce a denser representation when compared to the input PC, whose retained fine detailed features such as those in pedestrians and cyclists seem to aid the detection methods. The differences between MJ L-PCC and the better performing MJ LC-PCC can be attributed to the denser cylindrical representation shown in Fig. 5—Density Metric D3.

Comparing DL with non-DL compression methods, the 3D object detection performance remained comparable for similar objective quality of the reconstructed PCs.

5.2 Compression effects on 3D object detection using re-trained models

In Fig. 11, we present the detection performance of the re-trained detection methods using reconstructed PCs. These models were evaluated on PCs with the same compression parameters as those used during training. The performance of all object detection methods improved for low bit rates when compared to the pre-trained evaluation in Fig. 10.

At an average coding rate of 1 bpp, there is a noticeable improvement in the 3D AP value for "Car" in the "Easy" category when using G-PCC compression. Specifically, PV-RCNN shows an increase from about 60% to above 80%, while PointPillars also experiences a similar behaviour from approximately 50% to above 80%. However, the improvement for SECOND and PointRCNN at this average coding rate is relatively small, as their initial performance with evaluation on pre-trained models was already around 75%. On the other hand, when PCs are compressed with L3C2 at an average coding rate of around 0.5 bpp, there is a consistent enhancement in the 3D AP value for "Car" across all detection methods, reaching 70%, except for PV-RCNN, which achieves an even higher detection rate of 80% in the "Easy" category.

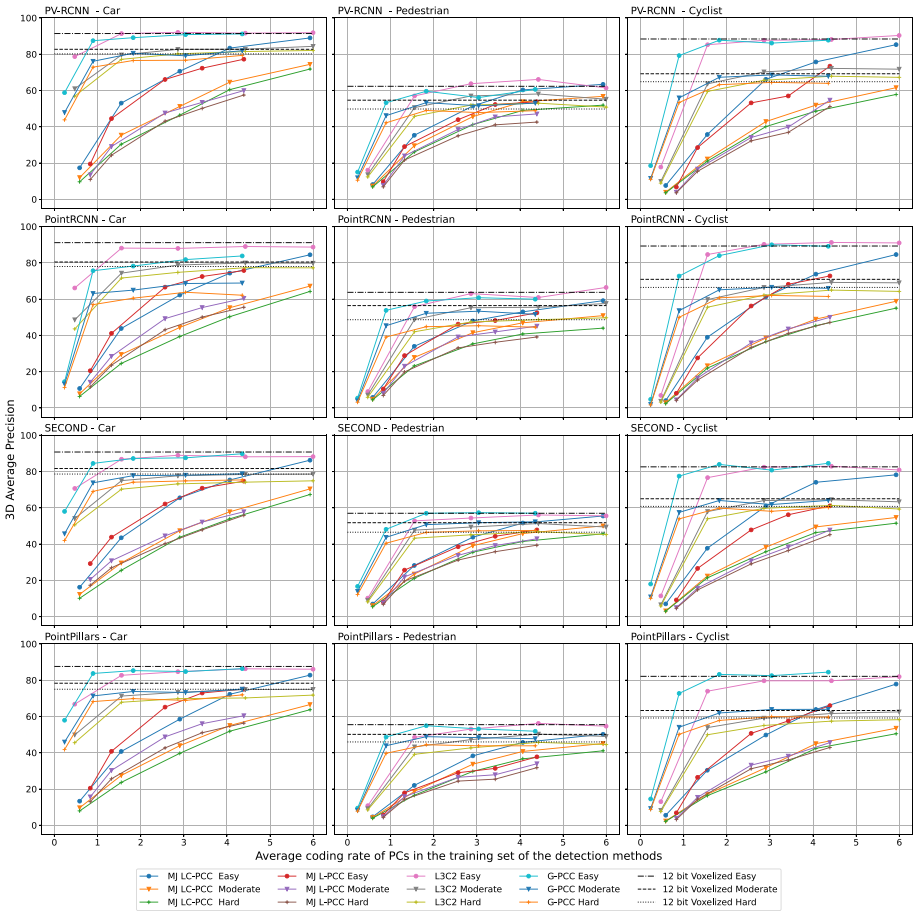


Fig. 11 3D average precision results for each object class and detection method evaluated on specifically trained models for each coding method and rate—method and class on top of each plot

MJ L-PCC and MJ LC-PCC also benefit from re-training, giving the detection methods a chance to learn from reconstructed PCs with a denser representation than that of the original KITTI dataset.

From the overall results in Fig. 11, there are significant improvements on detection performance when re-training the models with reconstructed PCs, relative to the pre-trained model performance. For the higher compression rates used, in the range of 2 bpp–6 bpp, it approaches the detection performance when using pre-trained models on the uncompressed data. In all cases, PCC below an average coding rate of 1 bpp degrades the PCs to the point where models also struggle to learn features that define an object class.

A special case can be observed with PointRCNN, that sees no meaningful improvements with re-training using compressed PCs, meaning that this detection model already learns to use PC features that are preserved during compression and re-training does not help to improve its performance. Interestingly, for PCs encoded with G-PCC, the detection performance actually worsens with re-training, particularly for the "Car" category. This can be attributed to the nature of the re-training process, which involves multiple epochs using data similar to the original, especially in cases where reconstructed PCs were subjected to lower compression ratios. Consequently, re-training may not necessarily lead to an improvement in detection performance and may introduce instability as the network struggles to converge towards a clear direction.

Also in this case, for both DL and non-DL compression methods, the 3D object detection performance using re-trained models remained comparable for similar objective quality of the reconstructed PCs.

In Figure 12, we present the evaluation result of the pre-trained PointRCNN model on a 12-bit voxelized PC before compression. This serves as the baseline for evaluating the performance of the re-trained models on the same reconstructed PC. This visual evaluation of the impact of compression on object detection is presented in Figure 13. Images on the left (13a, c, e, g) represent the scenario with lower compression, where compared to the baseline, object detection performance is similar, as shown by the bounding boxes around detected objects. Conversely, the images on the right (13b, d, f, h) demonstrate the result for higher compression. In this case, there is a noticeable degradation in object detection performance, where bounding boxes appear misaligned, there are false-positive classifications, and objects are not detected altogether when compared to the

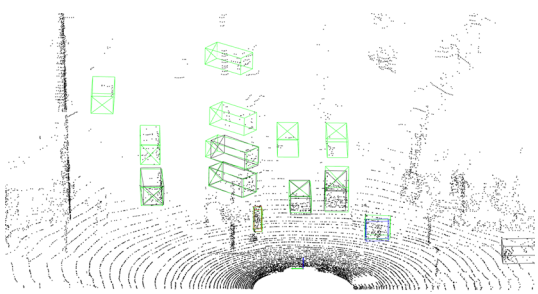
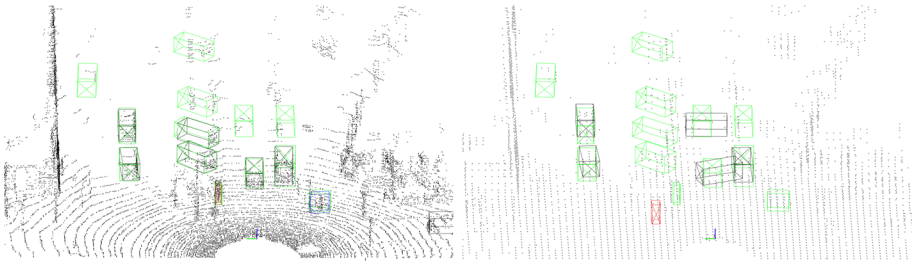
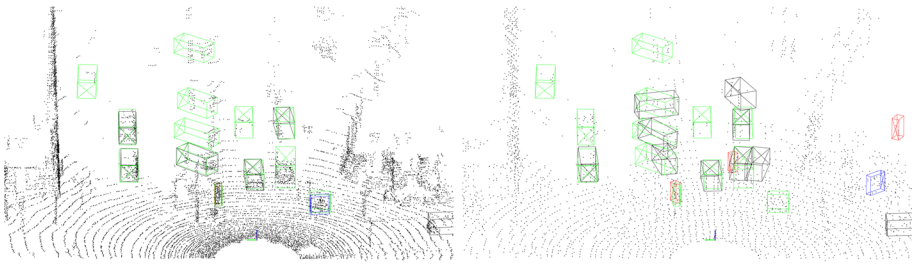


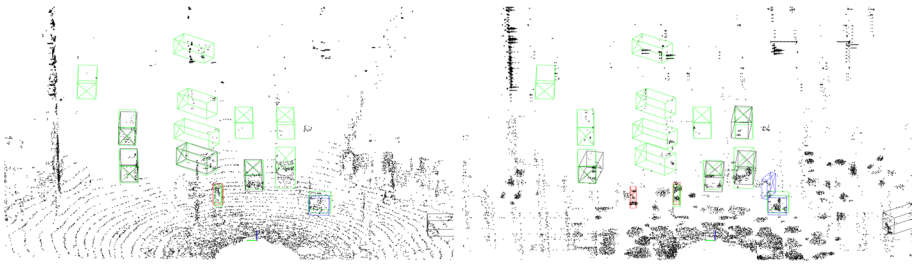
Fig. 12 Detection results using the pre-trained model of PointRCNN on 12-bit voxelized PC "003237". Light green boxes denote ground truth annotations, while black, red, and blue boxes represent detected instances of "Car", "Pedestrian" and "Cyclist" classes, respectively



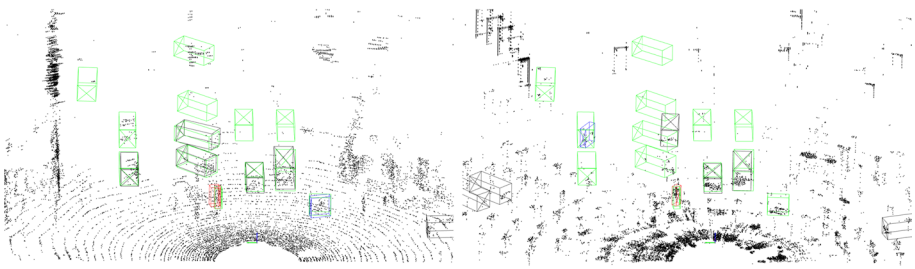
(a) G-PCC at R04 (4.63 bpp) (b) G-PCC at R01 (0.43 bpp)



(c) L3C2 at R04 (6.19 bpp) (d) L3C2 at R01 (0.78 bpp)



(e) MJ L-PCC at $\lambda = 0.00005$ (4.56 bpp) (f) MJ L-PCC at $\lambda = 0.0006$ (0.97 bpp)



(g) MJ LC-PCC at $\lambda = 0.00005$ (2.86 bpp) (h) MJ LC-PCC at $\lambda = 0.0006$ (0.34 bpp)

Fig. 13 Detection results using re-trained models of PointRCNN on PC "003237" compressed with G-PCC, L3C2, MJ L-PCC and MJ LC-PCC. Light green boxes denote ground truth annotations, while black, red, and blue boxes represent detected instances of "Car", "Pedestrian" and "Cyclist" classes, respectively

baseline, indicating instances where the compressed data adversely affected the accuracy of object detection.

6 Conclusions

Our investigation into PCC and object detection using various methods, has produced valuable conclusions into both PC coding and object detection performance, as well as on the joint effects of compressing the source PCs on object detection performance. The successful adaptation of JPEG PCC for LiDAR, specifically for Cartesian (MJ L-PCC)

and cylindrical (MJ LC-PCC) coordinates provided the means to include DL-based coding methods into this study. Both MJ L-PCC and MJ LC-PCC are reliant on a voxel occupancy probability threshold and the partition of PCs into blocks for encoding. This results in increased point density and artefact generation on the borders of the partition blocks, impacting objective quality and reconstruction consistency.

In what concerns PC coding alone, comparing the RD performance, non-DL methods such as G-PCC and L3C2 consistently outperformed MJ L-PCC and MJ LC-PCC methods, with L3C2 demonstrating high consistency in reconstructed representations across different scenes. G-PCC in angular mode, while better on average, exhibited most variations in RD depending on the scene environment.

In what concerns object detection performance using pre-trained models on compressed PCs, degradation was observed across all tested techniques and difficulty levels. G-PCC outperformed L3C2 for object detection, aligning with their respective RD results and showing that the smaller variation in objective quality in reconstruction of L3C2 does not contribute to better detection performance. It is only for the case of "Pedestrian" that the DL-based methods, MJ L-PCC and MJ LC-PCC, achieve detection performance close to that of G-PCC and L3C2, for the tested compression ratios. Furthermore, the use of cylindrical coordinates improves the detection performance of the DL-based detection methods, with MJ LC-PCC outperforming MJ L-PCC, independently of the used detection model for the case of "Pedestrian" and "Cyclist" classes.

In terms of object detection performance in a re-training scenario, object detection methods demonstrated improved performance at the higher compression ratios, that is, in the lower rate range. PV-RCNN, SECOND and PointPillars improved over the pre-trained model detection for average coding rate values below 3 bpp, benefiting from learning on reconstructed PCs.

Overall, we can conclude that PCs can be compressed using G-PCC or L3C2, down to 6 bpp while maintaining minimal detection degradation with pre-trained detection models. This is especially important when taking into account the original structure of the tested KITTI dataset that represents each PC coordinate component as a 4-byte floating-point number. This highlights the potential for achieving better storage and transmission efficiency by employing compressed representations of PCs for object detection purposes.

Furthermore, using pre-trained detection models makes compression factors as low as 4 bpp feasible, provided that a certain level of detection performance degradation is acceptable for the intended application. In scenarios where preserving higher detection performance is crucial at average compression rates below this threshold, using re-trained models proves to be effective in improving detection capabilities down to compression rates as low as 2 bpp. Also, while the used DL-based methods exhibit lower compression efficiency in this context, the impact on 3D object detection performance remains comparable across both DL and non-DL methods for similar values of objective quality.

Future research should explore alternatives to the partition-based DL compression methods. One path worth investigating is the adoption of a sparse tensor-based representation, which relies solely on occupied voxel information, eliminating the requirement for PC partitioning.

Acknowledgements
Not applicable.

Author contributions
Nuno A. B. Martins developed the adaptation of the DL-based compression method to accept LiDAR data. He also made data preparation, test, and analysis of the compression methods and 3D object detection performance, as well as, the manuscript preparation. Luís A. da Silva Cruz and Fernando Lopes participated in the development of the presented experiments. All authors read and approved the final manuscript.

Funding
This research was supported by the Instituto de Telecomunicações under Fundação para a Ciência e Tecnologia (FCT) projects UIDB/EEA/50008/2020 (https://doi.org/10.54499/UIDB/50008/2020) and LA/P/0109/2020, and, FCT doctoral grant 2023.00308.BDANA.

Availability of data and materials
The datasets generated and/or analyzed during the current study are available on request from the corresponding author.

Declarations

Competing interests
Luís A. da Silva Cruz is a Guest Editor for this publication.

Received: 8 February 2024 Accepted: 9 June 2024
Published online: 17 June 2024

References

1. S. Schwarz, M. Preda, V. Baroncini, M. Budagavi, P. Cesar, P.A. Chou et al., Emerging MPEG standards for point cloud compression. IEEE J. Emerg. Sel. Top. Circuits Syst. **9**(1), 133–148 (2019). https://doi.org/10.1109/JETCAS.2018.2885981
2. C. Tu, E. Takeuchi, A. Carballo, K. Takeda, Real-time streaming point cloud compression for 3D LiDAR sensor using U-Net. IEEE Access. **7**, 113616–113625 (2019). https://doi.org/10.1109/access.2019.2935253
3. T. Gandor, J. Nalepa, First gradually, then suddenly: understanding the impact of image compression on object detection using deep learning. Sensors. (2022). https://doi.org/10.3390/s22031104
4. L. Garrote, J. Perdiz, L.A. da Silva Cruz, U.J. Nunes, Point cloud compression: impact on object detection in outdoor contexts. Sensors. **22**(15), 1–12 (2022). https://doi.org/10.3390/s22155767
5. C.R. Qi, L. Yi, H. Su, L.J. Guibas, PointNet++: Deep hierarchical feature learning on point sets in a metric space. Advances in Neural Information Processing Systems. 2017:5100–5109. (2017) arXiv:1706.02413
6. A. Geiger, P. Lenz, R. Urtasun, Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In: Conference on Computer Vision and Pattern Recognition (CVPR); (2012)
7. R. Schnabel, R. Klein, Octree-based Point-Cloud Compression. Symposium on Point-Based Graphics 2006. (2006)
8. Y. Huang, J. Peng, C. Kuo, M. Gopi, Octree-Based Progressive Geometry Coding of Point Clouds. Eurographics Symposium on Point-Based Graphics (SPBG). (2006). https://doi.org/10.2312/SPBG/SPBG06/103-110
9. J. Kammerl, N. Blodow, R.B. Rusu, S. Gedikli, M. Beetz, E. Steinbach, Real-time compression of point cloud streams. Proceedings - IEEE International Conference on Robotics and Automation. p. 778–785. (2012). https://doi.org/10.1109/ICRA.2012.6224647
10. D.C. Garcia, R. L. De Queiroz, Context-based octree coding for point-cloud video. Proceedings - International Conference on Image Processing, ICIP. 2017-Septe(September):1412–1416, (2018). https://doi.org/10.1109/ICIP.2017.8296514
11. D. Thanou, P.A. Chou, P. Frossard, Graph-based compression of dynamic 3D point cloud sequences. IEEE Trans. Image Process. **25**(4), 1765–1778 (2016). https://doi.org/10.1109/TIP.2016.2529506. arXiv:1506.06096
12. B. Kathariya, L. Li, Z. Li, J. Alvarez, J. Chen, Scalable Point Cloud Geometry Coding with Binary Tree Embedded Quadtree. Proceedings - IEEE International Conference on Multimedia and Expo. 2018-July. (2018). https://doi.org/10.1109/ICME.2018.8486481
13. T. Golla, R. Klein, Real-time point cloud compression. IEEE International Conference on Intelligent Robots and Systems. 2015-Decem:5087–5092. (2015). https://doi.org/10.1109/IROS.2015.7354093
14. T. Ochotta, D. Saupe, Compression of point-based 3D models by shape-adaptive wavelet coding of multi-height fields. Proc Symposium on Point-Based Graphics. p. 103–112. (2004)
15. T. Ochotta, D. Saupe, Image-based surface compression. Comput. Graphics Forum. **27**(6), 1647–1663 (2008). https://doi.org/10.1111/j.1467-8659.2008.01178.x
16. E. Hubo, T. Mertens, T. Haber, P. Bekaert, Self-similarity based compression of point set surfaces with application to ray tracing. Comput. Graph. (Pergamon). **32**(2), 221–234 (2008). https://doi.org/10.1016/j.cag.2008.01.012
17. P.A. Chou, M. Koroteev, M. Krivokuca, A volumetric approach to point cloud compression–part I: attribute compression. IEEE Trans. Image Process. **29**(c), 2203–2216 (2020). https://doi.org/10.1109/TIP.2019.2908095
18. M. Krivokuca, P.A. Chou, M. Koroteev, A volumetric approach to point cloud compression-Part II: geometry compression. IEEE Trans. Image Process. **29**(c), 2217–2229 (2020). https://doi.org/10.1109/TIP.2019.2957853

19. D. Graziosi, O. Nakagami, S. Kuma, A. Zaghetto, T. Suzuki, A. Tabatabai, An overview of ongoing point cloud compression standardization activities : video- based (V-PCC) and geometry-based (G-PCC). APSIPA Trans. Sig. Info. Process. **9**, E13 (2020). https://doi.org/10.1017/ATSIP.2020.12
20. G-PCC Codec Description V12. In: MPEG Output Document N00151 (ISO). ISO/IEC JTC 1/SC 29/WG 7; (2021)
21. S. LASSERRE, J. TAQUET, A point cloud codec for Lidar data with very low complexity and latency. In: m56477. ISO/IEC JTC 1/SC 29/WG 7; (2021)
22. C. Tu, E. Takeuchi, C. Miyajima, K. Takeda, Compressing continuous point cloud data using image compression methods. IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC. p. 1712–1719, (2016). https://doi.org/10.1109/ITSC.2016.7795789
23. K. Kohira, H. Masuda, POINT-CLOUD COMPRESSION for VEHICLE-BASED MOBILE MAPPING SYSTEMS USING PORTABLE NETWORK GRAPHICS. ISPRS Ann. Photogramm. Remote Sens. Spat. Info. Sci. **4**(2W4), 99–106 (2017). https://doi.org/10.5194/isprs-annals-IV-2-W4-99-2017
24. C. Tu, E. Takeuchi, C. Miyajima, K. Takeda, Continuous point cloud data compression using SLAM based prediction. IEEE Intelligent Vehicles Symposium, Proceedings. (Iv):1744–1751, (2017). https://doi.org/10.1109/IVS.2017.7995959
25. H. Yin, C. Berger, Mastering data complexity for autonomous driving with adaptive point clouds for urban environments. IEEE Intelligent Vehicles Symposium, Proceedings. (Iv):1364–1371, (2017). https://doi.org/10.1109/IVS.2017.7995901
26. M. Isenburg, LASzip: lossless compression of lidar data. Photogramm. Eng. Remote. Sens. **79**, 209–217 (2013). https://doi.org/10.14358/PERS.79.2.209
27. M.Quach, G. Valenzise, F. Dufaux, Learning Convolutional Transforms for Lossy Point Cloud Geometry Compression. Proceedings - International Conference on Image Processing, ICIP. 2019-Septe:4320–4324, (2019). arXiv:1903.08548
28. M. Quach, G. Valenzise, F. Dufaux, Improved Deep Point Cloud Geometry Compression. IEEE 22nd International Workshop on Multimedia Signal Processing, MMSP 2020. (2020). arXiv:2006.09043
29. M. Quach, G. Valenzise, F. Dufaux, Folding-Based Compression of Point Cloud Attributes. Proceedings - International Conference on Image Processing, ICIP. 2020-Octob:3309–3313. (2020). arXiv:2002.04439
30. J. Wang, H. Zhu, H. Liu, Z. Ma, S. Member, Lossy point cloud geometry compression via end-to-end learning. **8215**(c), 1–15 (2021). https://doi.org/10.1109/TCSVT.2021.3051377
31. J. Wang, D. Ding, Z. Li, Z. Ma, Multiscale point cloud geometry compression. (Dcc):73–82, (2021). https://doi.org/10.1109/DCC50243.2021.00015
32. J. Wang, D. Ding, Z. Li, X. Feng, C. Cao, Z. Ma, Sparse tensor-based multiscale representation for point cloud geometry compression. IEEE Trans. Pattern Anal. Mach. Intell. **45**(7), 9055–9071 (2023). https://doi.org/10.1109/TPAMI.2022.3225816. arXiv:2111.10633v2
33. Y. Gao, P. Zhang, X. Wang, LOSSY LIDAR POINT CLOUD COMPRESSION VIA CYLINDRICAL 3D CONVOLUTION NETWORKS. IEEE Int. Conf. Image Process. (ICIP). **2023**, 3508–3512 (2023). https://doi.org/10.1109/ICIP49359.2023.10222471
34. C. Tu, E. Takeuchi, A. Carballo, K. Takeda, Point cloud compression for 3d lidar sensor using recurrent neural network with residual blocks. Proceedings - IEEE International Conference on Robotics and Automation. 2019-May:3274–3280, (2019). https://doi.org/10.1109/ICRA.2019.8794264
35. O. Ronneberger, P. Fischer, T. Brox, U-Net: convolutional networks for biomedical image segmentation. 2022 IEEE/CVF WACV **9**, 1989–1998 (2015). https://doi.org/10.48550/arXiv.1505.04597. arXiv:1505.04597
36. L. Wiesmann, A. Milioto, X. Chen, C. Stachniss, J. Behley, Deep compression for dense point cloud maps. IEEE Robot. Automat. Lett. **6**(2), 2060–2067 (2021). https://doi.org/10.1109/LRA.2021.3059633
37. H. Thomas, C.R. Qi, J.E. Deschaud, B. Marcotegui, F. Goulette, L. Guibas, KPConv: Flexible and deformable convolution for point clouds. Proceedings of the IEEE International Conference on Computer Vision. 2019-Octob:6410–6419, (2019). arXiv:1904.08889
38. ISO/IEC JTC 1/SC 29/WG1 N100097. Final Call for Proposals on JPEG Pleno Point Cloud Coding. 94th Meeting, Online, Jan 2022. (January):1–14, (2022)
39. ISO/IEC JTC1/SC29/WG1 N100367.: Verification Model Description for JPEG Pleno Learning-based Point Cloud Coding v1.0. https://ds.jpeg.org/documents/jpegpleno/wg1n100367-097-PCQ-Verification_Model_Description_for_JPEG_Pleno_Learning-based_Point_Cloud_Coding_v1_0.pdf
40. Davi, Lazzarotto, E. Touradj, Evaluating the effect of sparse convolutions on point cloud compression. In: 2023 11th European Workshop on Visual Information Processing (EUVIP). IEEE; (2023)
41. J. Mao, S. Shi, X. Wang, H. Li, 3D Object Detection for Autonomous Driving: A Review and New Outlooks. (2022); arXiv:2206.09474
42. C.R. Qi, H. Su, K. Mo, L.J. Guibas, PointNet: Deep learning on point sets for 3D classification and segmentation. Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017. 2017-Janua:77–85, (2017). arXiv:1612.00593
43. S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, et al., PV-RCNN: point-voxel feature set abstraction for 3D object detection. (2019); arXiv:1912.13192
44. H. Wu, J. Deng, C. Wen, X. Li, C. Wang, J. Li, CasA: a cascade attention network for 3D object detection from LiDAR point clouds. IEEE Trans. Geosci. Remote Sens. (2022). https://doi.org/10.1109/TGRS.2022.3203163
45. Y. Zhou, O. Tuzel, VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. p. 4490–4499, (2018). arXiv:1711.06396
46. Y. Yan, Y. Mao, B. Li, Second: Sparsely embedded convolutional detection. Sensors (2018). https://doi.org/10.3390/s18103337
47. A.H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, O. Beijbom, Pointpillars: fast encoders for object detection from point clouds. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 2019:12689–12697, (2019). arXiv:1812.05784
48. X. Zhu, H. Zhou, T. Wang, F. Hong, W. Li, Y. Ma, et al., Cylindrical and Asymmetrical 3D Convolution Networks for LiDAR-based Perception. IEEE Transactions on Pattern Analysis and Machine Intelligence. (2021). arXiv:2109.05441

49. Y. Wang, A. Fathi, A. Kundu, D.A. Ross, C. Pantofaru, T. Funkhouser, et al., Pillar-based object detection for autonomous driving. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 12367 LNCS:18–34, (2020). arXiv:2007.10323
50. L. Fan, Z. Pang, T. Zhang, Y.X. Wang, H. Zhao, F. Wang, et al., Embracing single stride 3D object detector with sparse transformer. p. 8448–8458, (2022). arXiv:2112.06375
51. S. Shi, L. Jiang, J. Deng, Z. Wang, C. Guo, J. Shi, et al., PV-RCNN++: Point-voxel feature set abstraction with local vector representation for 3D object detection. (2021); arXiv:2102.00463
52. C. Szegedy, S. Ioffe, V. Vanhoucke, A.A. Alemi, Inception-v4, inception-ResNet and the impact of residual connections on learning. In: 31st AAAI Conference on Artificial Intelligence, AAAI 2017. p. 4278–4284, (2017)
53. G-PCC test model v14. ISO/IEC JTC 1/SC 29/WG 7, Doc N00094. Online - April 2017;p. 19
54. ISO/IEC JTC 1/SC 29/WG 11.: Common test conditions for point cloud compression
55. A. Zaghetto, D Graziosi, A. Tabatabai, On density-to-density distortion; Technical Report, ISO/IEC JTC1/SC29/WG7 m60331. ISO/IEC
56. D. Tian, H. Ochimizu, C. Feng, R. Cohen, A. Vetro, Updates and integration of evaluation metric software for PCC. Technical Report M40522, ISO/IEC JTC1/SC29/WG11(MPEG)
57. X. Chen, K. Kundu, Y. Zhu, A. Berneshawi, H. Ma, S. Fidler, et al., 3D object proposals for accurate object class detection. Adv Neural Info Process Syst. 2015:424–432, (2015)
58. OpenPCDet Development Team.: OpenPCDet: An open-source toolbox for 3D object detection from point clouds. https://github.com/open-mmlab/OpenPCDet

Publisher's Note