IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 17 July 2022, accepted 1 August 2022, date of publication 8 August 2022, date of current version 12 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3197295

RESEARCH ARTICLE

Lossless Compression of Point Cloud Sequences Using Sequence Optimized CNN Models

EMRE C. KAYA AND IOAN TABUS, (Senior Member, IEEE)
Computing Sciences Unit, Tampere University, 33720 Tampere, Finland
Corresponding author: Emre C. Kaya (emre.kaya@tuni.fi)

ABSTRACT In this paper we propose a new paradigm for encoding the geometry of dense point cloud sequences, where a convolutional neural network (CNN), which estimates the encoding distributions, is optimized on several frames of the sequence to be compressed. We adopt lightweight CNN structures, we perform training as part of the encoding process and the CNN parameters are transmitted as part of the bitstream. The newly proposed encoding scheme operates on the octree representation for each point cloud, consecutively encoding each octree resolution level. At every octree resolution level, the voxel grid is traversed section-by-section (each section being perpendicular to a selected coordinate axis), and in each section, the occupancies of groups of two-by-two voxels are encoded at once in a single arithmetic coding operation. A context for the conditional encoding distribution is defined for each two-by-two group of voxels based on the information available about the occupancy of the neighboring voxels in the current and lower resolution layers of the octree. The CNN estimates the probability mass functions of the occupancy patterns of all the voxel groups from one section in four phases. In each new phase, the contexts are updated with the occupancies encoded in the previous phase, and each phase estimates the probabilities in parallel, providing a reasonable trade-off between the parallelism of the processing and the informativeness of the contexts. The CNN training time is comparable to the time spent in the remaining encoding steps, leading to competitive overall encoding times. The bitrates and encoding-decoding times compare favorably with those of recently published compression schemes.

INDEX TERMS Convolutional neural networks, lossless geometry compression, octree coding, point cloud compression.

I. INTRODUCTION

The compression of the voxelized point clouds recently became a hot research topic, owing to the need to develop immersive technologies, underlined, e.g., in the programs launched by the MPEG [1] and JPEG [2] standardization bodies, which have already resulted in two well-engineered standards, V-PCC [3] and G-PCC [4] (having the test model TMC13 [5]). The scientific literature has witnessed a strong interest in improving the compression performance of G-PCC, with many scholarly contributions in recent years, e.g., [6]–[22]. These methods differ in several aspects, such as the representation used for the point cloud (e.g., octree [15]–[20], dyadic decomposition [6], [7], [12] or projections onto 2D planes [13], [23]), the selection of the context used for the conditional probability model for arithmetic coding and the method to define the symbols to be encoded.

The probability model can be based on adaptively maintained counts for various contexts [4], [7] or on a neural network (NN) model [11], [17], having a binary context at the input and the probability mass function of the symbol at the output.

In the last few years, machine learning approaches using neural networks have been proven to be competitive for both lossy [14], [16], [20]–[22] and lossless [11], [17], [20], [24] point cloud geometry compression. A comprehensive survey of the recent methods with a focus on the learning-based approaches is provided in [25].

In [14], an autoencoder architecture involving 3D convolutional layers is employed to generate a latent representation

The associate editor coordinating the review of this manuscript and approving it for publication was Gulistan Raja.

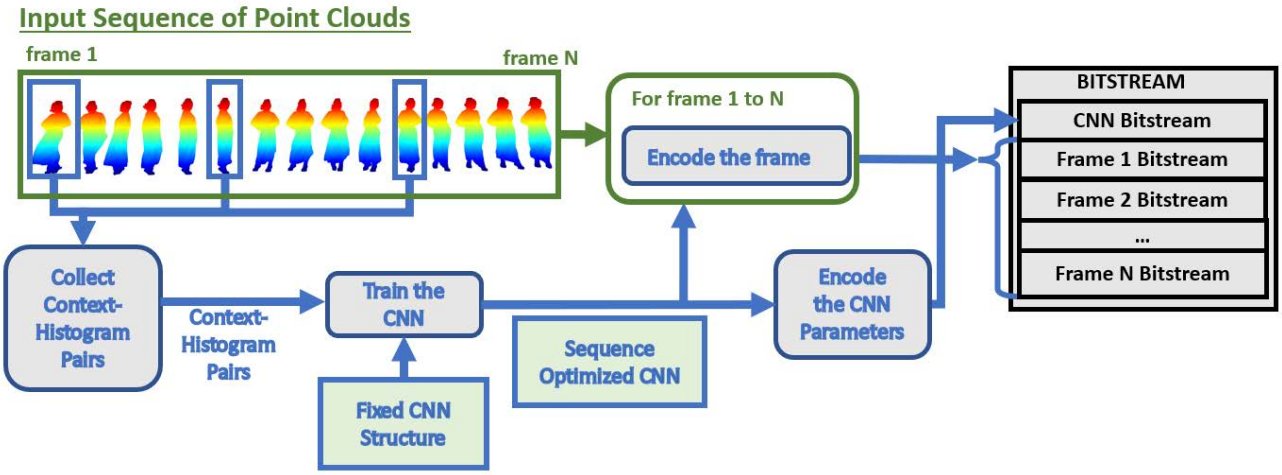

FIGURE 1. Overview of the proposed Specifically Trained Model (STM) compression method for encoding a point cloud sequence.

of the point cloud, which is further compressed using range coding. In [21], a variational autoencoder model is employed in an end-to-end learning scheme. In [22], adaptive octree-based decomposition of the point cloud is performed prior to encoding with a multilayer perceptron-based end-to-end learned analysis-synthesis architecture. VoxelDNN [11] uses 3D masked convolutional filters to enforce the causality of the 3D context from which the occupancy of $64 \times 64 \times 64$ blocks of voxels are estimated. VoxelContextNet [20] employs an octree-based deep entropy model for both dynamic and static LIDAR point clouds. NNOC [17] operates on the octree representation, where hybrid contexts are formed by combining the information from two consecutive resolution levels. In [24], a deep generative model is employed for lossless geometry compression.

As a variation from the previous approaches that utilize neural networks, we optimize a specific neural network model for the sequence to be encoded. In the proposed scheme, the optimization (training) of the network is a part of the encoding stage, and the optimized CNN model parameters are transmitted as a header to the decoder, being then used for decoding any point cloud from the sequence.

We consider here two distinct paradigms for using a neural network as a coding probability model. In the first, which we dub *Generically Trained Model* (GTM), some generic training set is selected and is used for optimizing an NN model to be used by both the encoder and decoder for all the compression tasks that will be required in the future [8], [11], [14], [16], [17], [20]–[22], [24], [26], [27]. The compression performance of the methods involving this approach depends on the selection of a suitable training set. The model is an integral part of the encoding program, and an identical copy is assumed to exist in the decoding program, so the NN model is not considered as a part of the encoded data. Hence, the model can be made as complex as needed, since its size does not contribute to the size of the bitstream. Additionally, the training time of the model is not accounted for in the encoding time, although it can be rather important, of the order of hours.

Here, we propose a second paradigm, *Specifically Trained Model* (STM), where the NN model to be used for the compression of a sequence of point clouds (PC) is optimized for the sequence and is transmitted as a part of the encoded stream. One cannot simply adhere to this paradigm by selecting the same model structure as in the GTM case and train it on the sequence, because the complexity-compression ratio trade-off needs to be different. The model needs to be trained quickly, and the cost of transmitting the model parameters needs to be sufficiently low. Using the specifically trained model paradigm as a basis for the compression of a point cloud sequence has the advantage that the NN model that generates the coding distribution at each context can be trained to match very closely the real distributions found in the point clouds that form the sequence, even with a less complex NN structure than in the general trained model case. Another advantage is that the point clouds in a sequence are similar, one to another, from the point of view of their real probability distributions at the contexts. Whereas in the case of a general trained model, the distributions generated by the model at each context can differ quite significantly from the distributions learned from the generic training data.

In this paper, we propose a solution that belongs to the STM paradigm and has the following main features: It uses the octree representation [28]; encodes the occupancy of groups of 2×2 voxels at each arithmetic encoding operation; uses a context based on the occupancies of lower resolution voxels (translated as candidate voxels at the current resolution) and on sets of voxels already encoded at the current resolution; computes the probability using a CNN model having at the input a multivalued context (which becomes gradually more relevant along the four phases).

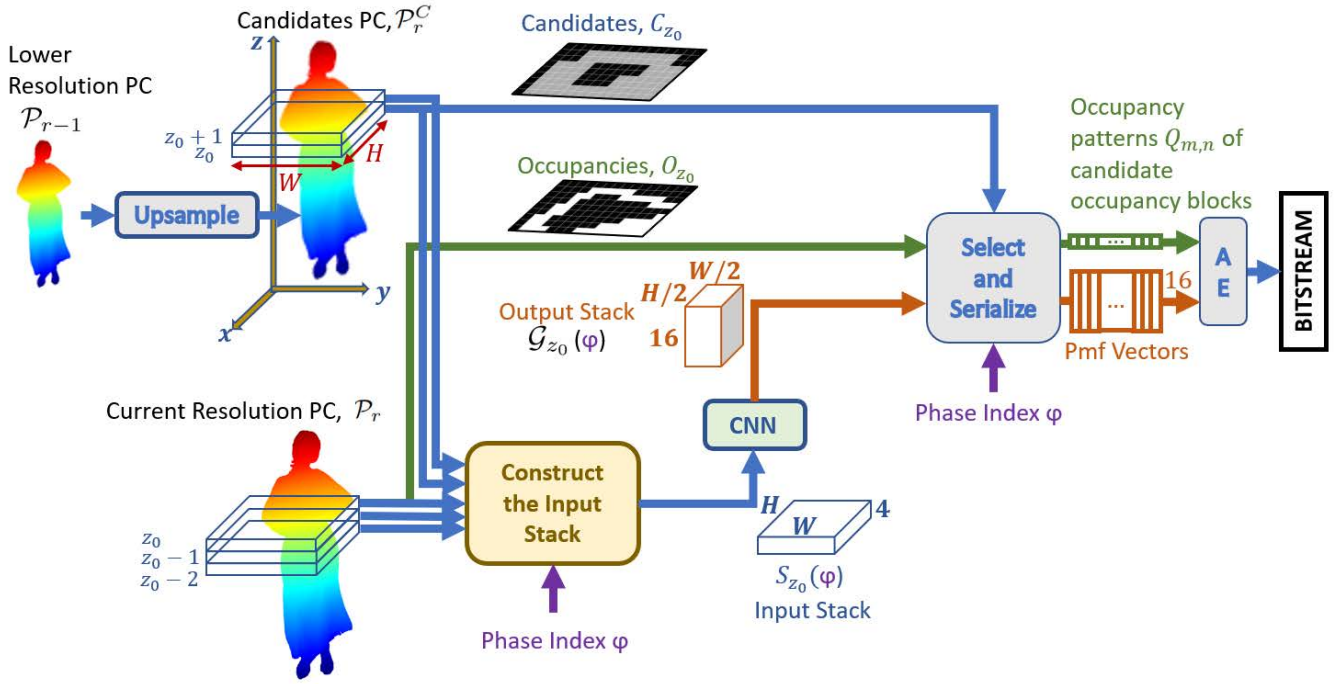

FIGURE 2. Encoding of the occupancy image O_{z_0} (the section $z = z_0$ through the point cloud $\mathcal{P}_r$ at resolution r) at phase φ using an arithmetic encoder (AE) for 16-valued symbols. "Construct the Input Stack" is detailed in Fig. 3, CNN is detailed in Fig. 4 and "Select and Serialize" is detailed in Fig. 5.

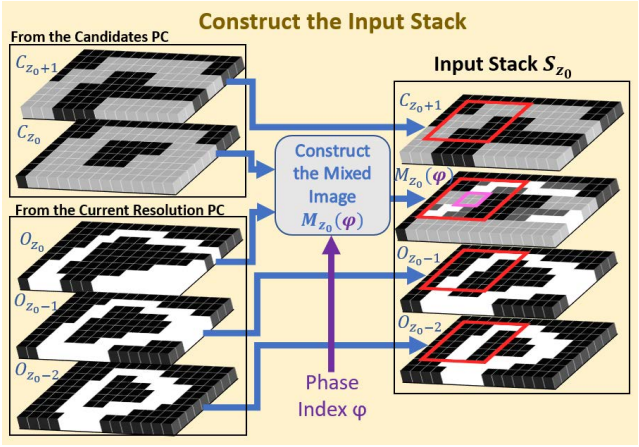

FIGURE 3. Construction of the Input Stack $S_{z_0}(\varphi)$. $M_{z_0}(\varphi)$ is formed by combining the available information from the current and the lower resolution (candidates PC). For more detail on how $M_{z_0}(\varphi)$ is formed, see Fig. 7 and Eqn. (3).

The structure of the paper is as follows. We introduce our method in Section II and present the experimental results in Section III, and we present the conclusions in Section IV.

II. PROPOSED METHOD

A. ENCODING A SEQUENCE OF FRAMES BY OPTIMIZING A CNN MODEL ON A FEW FRAMES

We consider lossless compression of the geometry of dense point clouds forming a sequence. Each point cloud in the sequence is referred to as a frame. An overview of the proposed lossless sequence encoding scheme is provided in Fig. 1. The lossless encoding scheme consists of three stages. In the first stage, we collect the contexts and corresponding histograms from a small number of frames. For example, we show in the Experimental Work section the performance when selecting different numbers of training frames at equal temporal distances to each other. More elaborate selection strategies might be considered but we noticed that the overall performance does not improve significantly for the sequences that we experiment with. Each context is associated with a 16-element histogram where each element corresponds to an occupancy pattern as described in Section II-C in detail. This first stage is shown as a block called "Collect Context-Histogram Pairs" in Fig. 1. In the second stage, we train a fixed structure CNN model using the contexts and corresponding histograms that were collected in the first stage (see "Train CNN" in Fig. 1). We emphasize that the structure of the CNN is fixed to make it clear that it is not subject to optimization in the algorithm, however it is chosen *a priori* from a set of possible structures, given the intended performance: for better compression performance one can use a heavier model, while for low overhead of the model bitstream a lighter structure is preferable (e.g., in the case of short sequences). In Section III we illustrate four choices of CNN model structures of different complexities and report their performance. The set of possible structure choices should be known to the decoder and the choice made at the encoder needs to be signaled (by a couple of bits) to the decoder.

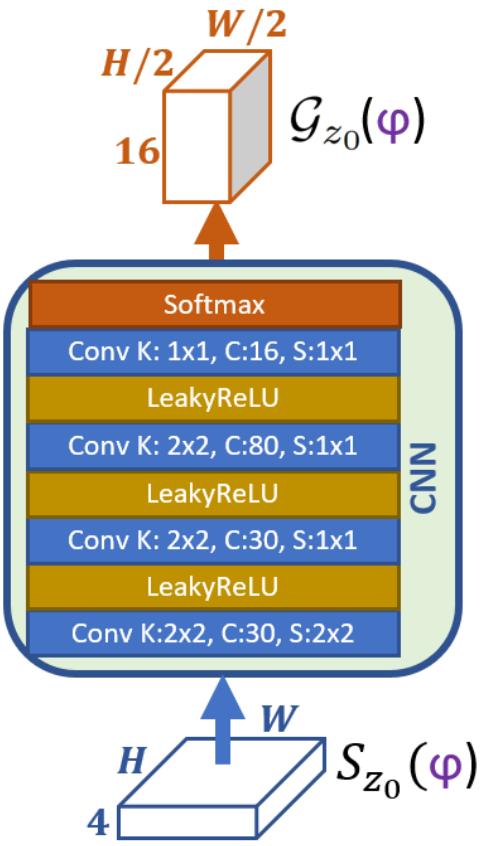

FIGURE 4. Structure of the sequence optimized CNN used for estimating the occupancy probabilities. For the convolutional layers, K denotes the 2D kernel size, C denotes the number of output channels and S denotes the 2D stride.

The optimal parameters (weights and biases) of the CNN resulting from the training stage, are losslessly transmitted, consuming 32 bits for each floating point parameter. This corresponds to the "Encode the CNN Parameters" block in Fig. 1. The decoder has available the structure parameters of the CNN (number of layers, number of channels, strides), and by reading the header transmitted by the encoder, it can reconstruct the CNN model to be exactly the same as the one used by the encoder. The CNN model is used for independently encoding/decoding each frame of the sequence, and since the encoding is done only intraframe, the starts and ends of the bitstream for each frame can be stored (incurring a little overhead) so that each frame can be decoded without the need of decoding the previous frames. In the final stage, we encode the entire sequence frame-by-frame using the same CNN for each frame. The frames can be encoded independently from one another; hence, they can be decoded in any order, independently from one another, resulting in random access to the point clouds of the sequence. We note that the methods using interframe coding do not possess this random access property.

B. ENCODING OF A FRAME

At each frame, the octree representation of the point cloud is processed iteratively at the resolution r, starting from the resolution (octree depth level) $r = 2$ and eventually reaching the original resolution $r = R$, thus enabling lossless reconstruction. First, the PC at $r = 2$ is written to the bitstream in 64 bits, where each bit represents the occupancy of a voxel.

The encoding steps at resolutions higher than 2 are illustrated in Fig. 2. Let $\mathcal{P}_R$ denote the input point cloud, where $\mathcal{P}_r$ with $2 \leq r < R$ is a lower resolution version of $\mathcal{P}_R$. At each resolution level $r \leq R$, we have available at both the encoder and the decoder the point cloud $\mathcal{P}_{r-1}$, and we create from it an upsampled version called $\mathcal{P}_r^C$ by splitting each occupied voxel of $\mathcal{P}_{r-1}$ into eight candidate voxels in $\mathcal{P}_r^C$, now having a resolution of r bits per dimension. The point cloud $\mathcal{P}_r$ is a subset of $\mathcal{P}_r^C$. For encoding $\mathcal{P}_r$, we need to encode and transmit the occupancy status of each candidate voxel in $\mathcal{P}_r^C$. This is performed sequentially by sweeping z_0 along the sweeping dimension z and at each z_0 considering the sectioning by the plane $z = z_0$ of the point cloud $\mathcal{P}_r^C$ and for each candidate voxel transmitting the occupancy status.

To use suggestive geometric interpretation, when encoding the voxels at the section $z = z_0$ in $\mathcal{P}_r$, we refer to the plane in the x and y coordinates as a binary $(W \times H)$-occupancy image O_{z_0}, where $O_{z_0}(x, y) = 1$ indicates that $(x, y, z_0) \in \mathcal{P}_r$ (see Fig. 2). The occupancies of the candidate voxels in the current section $z = z_0$ are encoded in four phases such that, at each phase, only some of the candidate voxels in the current section are encoded. The context used at a phase φ for encoding the candidate voxels is constructed using the most up-to-date information, containing the occupancies that are encoded in the previous phases $1, \ldots, \varphi - 1$. Thus, having four phases instead of a single phase provides more informative contexts for the candidates that are not encoded in the first phase.

Section $z = z_0$ through the upsampled candidate point cloud $\mathcal{P}_r^C$ is called the $(W \times H)$-candidate image, C_{z_0} (see Fig. 2). During the decoding process, the reconstructed occupancy image $R_{z_0}(\varphi)$ (a binary $(W \times H)$-image) is maintained for each of the four phases $\varphi = 1, 2, 3$, and 4, as described in Subsection II-C. After phases $\varphi = 1, 2, 3$, $R_{z_0}(\varphi)$ is a partial reconstruction of O_{z_0}, and after phase $\varphi = 4$, O_{z_0} is fully reconstructed. In $R_{z_0}(\varphi)$, the decoded occupied voxels (true points) are represented as pixels with a value of 1, and all the remaining pixels have a value of 0. When encoding the pixels of O_{z_0}, the planes O_{z_0-1} and O_{z_0-2} were already encoded, and we rely, when building conditioning distributions by the CNN, on the most relevant available information, namely, the images C_{z_0+1}, C_{z_0}, O_{z_0-1}, O_{z_0-2}, and for the information in the current plane $z = z_0$ reconstructed as $R_{z_0}(\varphi)$. The information from the candidate image C_{z_0} and the reconstructed image R_{z_0} are combined into a four-level mixed image, $M_{z_0}(\varphi)$, as described in Subsection II-C.

The mixed image $M_{z_0}(\varphi)$ together with the three binary images C_{z_0+1}, O_{z_0-1} and O_{z_0-2} are used to construct a $4 \times W \times H$ array called Input Stack $S_{z_0}(\varphi)$. Input Stack $S_{z_0}(\varphi)$ is constructed by the block "Construct the Input Stack" in Fig. 2, which is detailed in Fig. 3, and it is input to a CNN for estimating the probabilities of occupancy of the voxels at $z = z_0$.

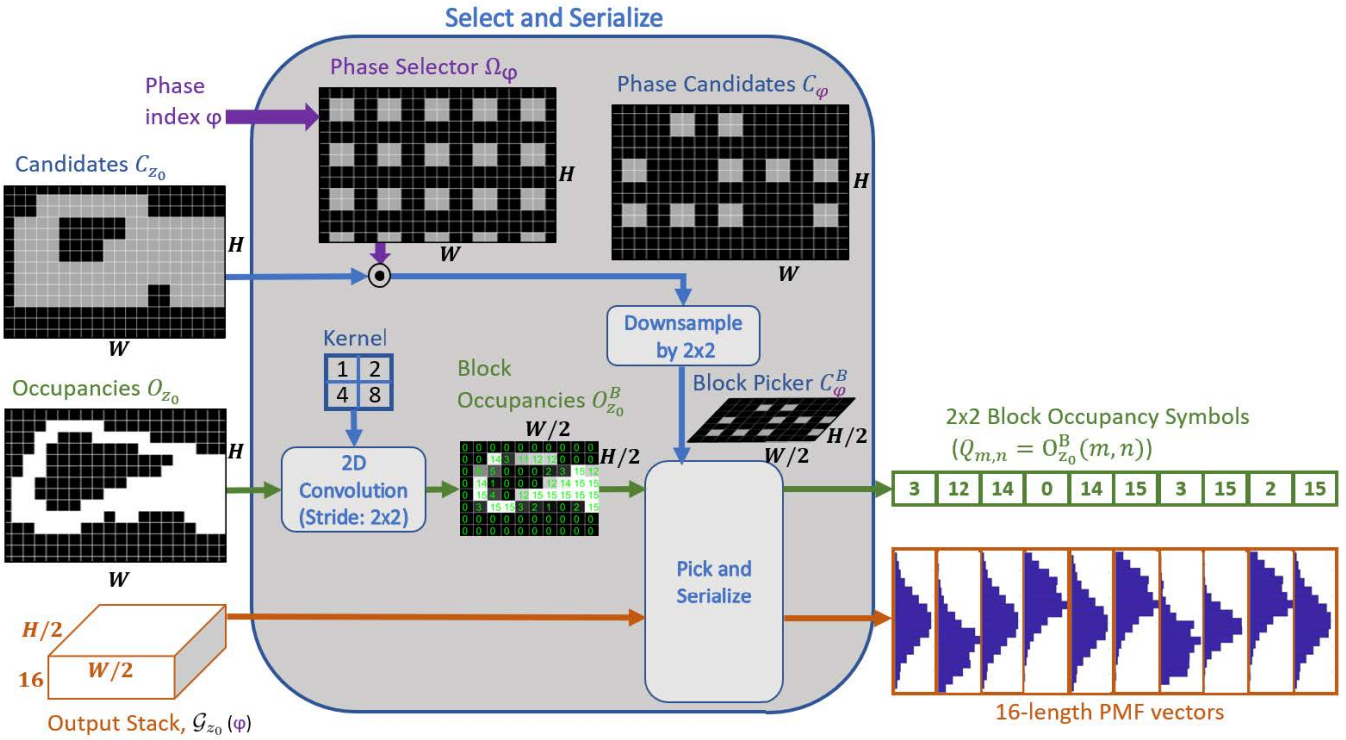

FIGURE 5. The structure of the block "Select and Serialize" from Fig. 2. The candidate voxels encoded in the current phase (φ) are determined by the phase candidate image C_φ, which is obtained by multiplying the two binary images C_{z_0} and Ω_φ. By downsampling C_φ, we obtain the so-called block picker C^B_φ, which is a binary image where $C^B_\varphi(m,n) = 1$ corresponds to a 2 × 2 block of occupancies $B_{m,n}$ that needs to be encoded in phase φ. Occupancies image O_{z_0} is convolved with a 2 × 2 kernel to generate the block occupancies image $O^B_{z_0}$, which contains numbers from 0 to 15 that symbolize the occupancy patterns of 2 × 2 blocks. Block Picker's shape is consistent with both $O^B_{z_0}$ and the output stack $\mathcal{G}_{z_0}(\varphi)$ coming from the CNN. The element $O^B_{z_0}(m,n)$ for which $C^B_\varphi(m,n) = 1$ and the corresponding column $\mathbf{G}(m,n)$ in the output stack $\mathcal{G}_{z_0}(\varphi)$ are picked and form a (symbol, pmf) pair. These pairs are serialized in a certain scanning order, which is also obeyed by the decoder.

We encode the occupancy of candidate voxels in section $z = z_0$ in blocks of 2 × 2 voxels, such that a block of occupancies at a position m, n is defined as

$$B_{m,n} = \begin{bmatrix} O_{z_0}(2m, 2n) & O_{z_0}(2m, 2n+1) \\ O_{z_0}(2m+1, 2n) & O_{z_0}(2m+1, 2n+1) \end{bmatrix}.$$

We define the occupancy pattern of a block $B_{m,n}$ as $Q_{m,n} = O_{z_0}(2m, 2n) + 2O_{z_0}(2m, 2n+1) + 2^2 O_{z_0}(2m+1, 2n) + 2^3 O_{z_0}(2m+1, 2n+1)$. The entire $W \times H$ image is covered by nonoverlapping blocks by setting $m = 0, 1, \ldots, W/2-1$ and $n = 0, 1, \ldots, H/2-1$ (W and H are enforced to be even).

For encoding with arithmetic coding the occupancy pattern $Q_{m,n}$ of a generic (2 × 2)− block of pixels $B_{m,n}$, we utilize a probability mass function (pmf), denoted as a 16-length vector $\mathbf{G}(m,n)$, specifying at element q the probability $G_q(m,n) = \text{Prob}(Q_{m,n} = q|\mathcal{C}_{m,n})$, for each $q \in \{0, \ldots, 15\}$, conditioned on a context $\mathcal{C}_{m,n}$ (see Fig. 6). We implement the probability model by a CNN with the set of parameters $\mathcal{W}$ and 16 output channels, which will output at every location (m,n) the pmf vector denoted

$$\mathbf{G}(m,n) = CNN_{\mathcal{W}}(\mathcal{C}_{m,n}) = \mathbf{g}(\mathcal{W}, \mathcal{C}_{m,n}). \quad (1)$$

Hence, the output of the CNN is an array of size $16 \times W/2 \times H/2$, called the output stack, denoted $\mathcal{G}_{z_0}$. At location (m,n) and channel q, $\mathcal{G}_{z_0}(q,m,n) = G_q(m,n)$.

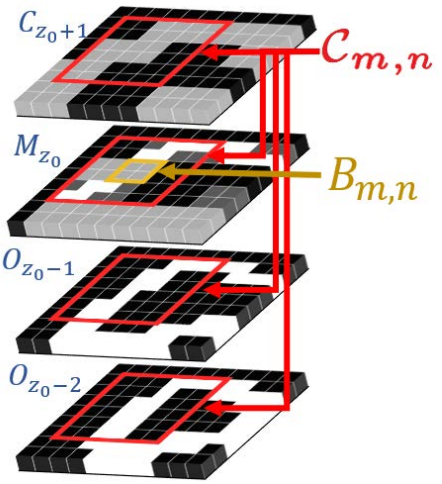

FIGURE 6. The context $\mathcal{C}_{m,n}$ (the 4 × 6x6 block of voxels marked in red) for computing the 16-element probability mass function $Prob(Q_{m,n} = q|\mathcal{C}_{m,n})$ for $q = 0, \ldots, 15$, needed for encoding and decoding the occupancy pattern $Q_{m,n}$ for the 2 × 2 block $B_{m,n}$ (the block marked by the yellow square). The CNN computes in parallel all these probabilities at all (m,n) locations. At the training stage, for each context observed in the training set, a 16-element histogram $h(q|\mathcal{C}_{m,n})$, $q = 0, \ldots, 15$, is collected and used in the loss function from (5).

The conditioning is done on a context $\mathcal{C}_{m,n}$, which is defined by the receptive field of the CNN, i.e., by the set of

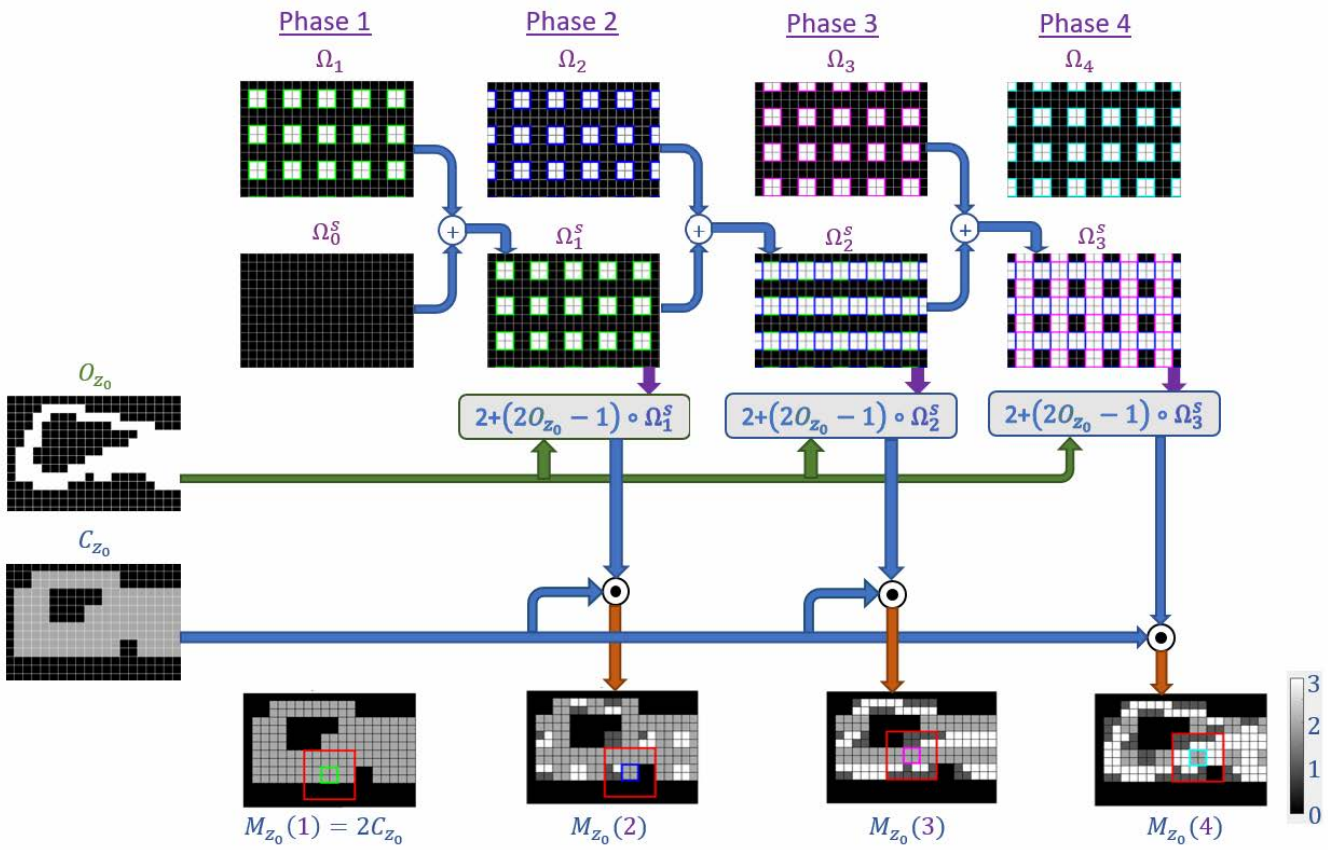

FIGURE 7. Constructing the mixed quaternary image $M_{z_0}(\varphi)$ at each phase φ by combining the available information from the already encoded parts of the binary occupancy image O_{z_0} and the binary candidates image C_{z_0}. The quaternary value for each pixel (i,j) is obtained by setting either $M_{z_0}(i,j)=2O_{z_0}(i,j)+1$ or $M_{z_0}(i,j)=2C_{z_0}(i,j)$ depending on whether $O_{z_0}(i,j)$ is known in the current phase (i.e., $\Omega^s_{\varphi-1}(i,j)=1$). Each phase is associated with a different phase selector image Ω_φ, which are binary images. The already processed voxels at the beginning of each phase φ are expressed as a binary image $\Omega^s_{\varphi-1}$. The mixed image, $M_{z_0}(\varphi)$, which is to be used in the input stack at phase φ, is obtained by applying $M_{z_0}(\varphi)=C_{z_0}\circ(2+(2O_{z_0}-1)\circ\Omega^s_{\varphi-1})$ (3) (which rephrases the mechanism of constructing the quarternary values described above). A pixel in $M_{z_0}(\varphi)$ can have one of the four values 0,1,2,3. If it was not yet encoded, it can be 0 (if it is not a candidate, $C_{z_0}(i,j)=0$) or 2 (if it is a candidate, $C_{z_0}(i,j)=1$). If it is already encoded, it can be 1 (if it is not occupied, $O_{z_0}(i,j)=0$) or 3 (if it is occupied, $O_{z_0}(i,j)=1$). At each phase, we show in $M_{z_0}(\varphi)$ one example context window as a red bounding box around its corresponding 2 × 2-block to be encoded.

pixels from the Input Stack S_{z_0}, which affects the computation of $\mathbf{G}(m,n)$. For the selected structure of the CNN shown in Fig. 4, the receptive field $\mathcal{C}_{m,n}$ can be seen to be the $4\times6\times6$ subblock from the input stack S_{z_0}, ranging on x coordinates from $2m-2$ to $2m+3$ and on y coordinates from $2n-2$ to $2n+3$. In Fig. 3 and Fig. 6, the context of $4\times6\times6$ pixels from the images $O_{z_0-2},\ldots,C_{z_0+1}$, corresponding to the $2\times2-$ candidate block marked in yellow on the mixed image $M_{z_0}(\varphi)$, are shown by the red contours.

The CNN consists of four 2D convolutional stages where the convolutions operate along the W and H dimensions and the number of input channels is four (the number of 2D images in $S_{z_0}(\varphi)$). As a nonlinear activation function at the hidden stages, we employ LeakyReLU [29] with a constant slope α for the negative inputs so that

$$\text{LeakyReLU}(x)=\max(0,x)+\alpha*\min(0,x). \qquad (2)$$

We set the negative region slope as $\alpha=0.01$ following [29]–[31]. The output layer activation is chosen as softmax to ensure the output $\mathbf{G}(m,n)$ to be valid probability mass functions.

C. ENCODING THE VOXELS IN A 2D SECTION IN FOUR PHASES

To improve the informativeness of the contexts, we split the encoding process into four phases such that in each phase, the CNN is called once, having at the input a different input stack, $S_{z_0}(\varphi)$, which gradually becomes more informative at each phase. In all four phases, we employ the same CNN model which is trained with context-histogram pairs collected from all four phases. We do not devote different CNN models to each phase because in such a scenario, the CNN bitstream would be four times longer and the CNN training time would also increase significantly. Moreover, running four different CNNs would significantly increase the consumption of GPU memory during encoding and decoding. Thus, having a single model which handles all the four phases is advantageous in terms of the bitrate, the execution times and the memory

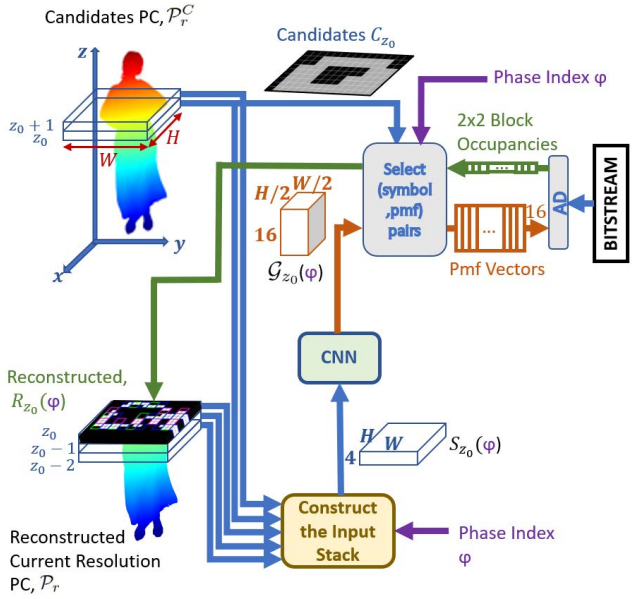

FIGURE 8. Decoding of the occupancy image O_{z_0} (the section $z = z_0$ through the point cloud $\mathcal{P}_r$ at resolution r) at a phase φ. The block "Select (symbol,pmf) pairs" provides the arithmetic decoder (AD) with the pmf vectors of the blocks to be decoded and it inserts the decoded block occupancies into the reconstructed $R_{z_0}(\varphi)$. In the next phase, $R_{z_0}(\varphi)$ is used to form the $M_{z_0}(\varphi+1)$ in the input stack $S_{z_0}(\varphi+1)$.

consumption. Associated with each phase φ is a different phase selector image Ω_φ that selects the candidate blocks to be encoded in the current phase (the selected candidate blocks are called the phase candidates blocks) (see Fig. 5 and 7).

Phase selector images Ω_φ are visualized in Fig. 7. In each phase φ, a quarter of the 2×2 blocks, all located in a squared grid, are selected for possible encoding by setting $\Omega_\varphi(i,j) = 1$ for every pixel (i,j) belonging to a selected 2×2 block (see the four phase selectors $\Omega_1, \ldots, \Omega_4$ at the top of Fig. 7). The elements of Ω_1 are 1 for all pairs (i,j) with $i = 4k, i = 4k+1, j = 4l, j = 4l+1$, where k, l are integers, so that the elementwise product $O_{z_0} \circ \Omega_1$ ($\circ$ denotes element- wise multiplication) forces to zero all the pixels that do not belong to blocks $B_{2k,2l}$.

Similarly, the selector image Ω_2 selects in $O_{z_0} \circ \Omega_2$ all the blocks $B_{2k,2l+1}$, Ω_3 selects all the blocks $B_{2k+1,2l}$, and Ω_4 selects all the blocks $B_{2k+1,2l+1}$.

In each phase, the CNN uses the Input Stack $S_{z_0}(\varphi)$ and generates pmf vectors $\mathbf{G}(m,n) = CNN_{\mathcal{W}}(\mathcal{C}_{m,n})$ for all blocks $B_{m,n}$, with $m = 0, 1, \ldots W/2 - 1$ and $n = 0, 1, \ldots H/2 - 1$, hence covering all blocks of the $(W \times H)$ images. The candidate image C_{z_0} specifies which of the blocks are already known to be zero and hence do not need to be encoded.

In the first phase, out of all $2 \times 2-$ blocks of C_{z_0} and O_{z_0}, only a quarter of the blocks are selected by using Ω_1; namely, from O_{z_0}, only the blocks $B_{2k,2l}$, with $k = 0, \ldots, W/4 - 1$ and $l = 0, \ldots, H/4 - 1$. The pmf corresponding to each block is read from the pmf vector $\mathbf{G}(2k, 2l)$ and is used by the arithmetic encoder. The encoder and the decoder are now accounting that one quarter of the image O_{z_0} was reconstructed, and those values can be inserted for the next phase into S_{z_0}.

Phases 2, 3 and 4 proceed in a similar way, after which the entire O_{z_0} pixels can be reconstructed so that $R_{z_0} = O_{z_0}$, and the algorithm moves to the processing of the next section, $z = z_0 + 1$.

Fig. 7 shows how the candidate image C_{z_0} and the occupancy image O_{z_0} (or equivalently the reconstructed image R_{z_0}) are combined differently at each phase, resulting in the mixed $W \times H$-image $M_{z_0}(\varphi)$. Initially, since from O_{z_0} nothing is encoded yet, M_{z_0} contains only the candidacy information. The already processed part of the current section after phase φ is denoted with a binary image $\Omega^s_\varphi = \Omega_1 \vee \ldots \vee \Omega_\varphi$ (elementwise OR), where Ω^s_0 is all-zeros. The mixed image $M_{z_0}(\varphi)$ at the beginning of phase φ is obtained at the encoder as

$$M_{z_0}(\varphi) = C_{z_0} \circ (2 + (2O_{z_0} - 1) \circ \Omega^s_{\varphi-1}), \tag{3}$$

where $\circ$ is pixelwise multiplication, by the process of obtaining the quaternary values out of the two binary values, as described in the caption of Fig. 7. Equivalently, at the decoder, we have

$$M_{z_0}(\varphi) = C_{z_0} \circ (2 + (2R_{z_0}(\varphi) - 1) \circ \Omega^s_{\varphi-1}), \tag{4}$$

since $O_{z_0} \circ \Omega^s_{\varphi-1} = R_{z_0}(\varphi) \circ \Omega^s_{\varphi-1}$. Eqn. (3) is the mathematical expression for the block "Construct the Mixed Image $M_{z_0}(\varphi)$" in Fig. 3, whereas (4) is the decoder version of the same block.

The "Select and Serialize" block, which appears in Fig. 2, carries out the operations required to feed the occupancy symbols and the corresponding probability mass functions to the arithmetic encoder. "Select and Serialize" operations are schematized in Fig. 5. The two binary images C_{z_0} and Ω_φ are element-wise multiplied to yield the so-called phase candidates C_φ. Occupancies image O_{z_0} is convolved with a 2D kernel having elements [1, 2; 4, 8] to yield the so-called block occupancies $O^B_{z_0}$. $O^B_{z_0}$ is a 16-level image with dimensions $(W/2, H/2)$, and each pixel corresponds to the occupancy pattern of a 2×2 block in O_{z_0}. In phase φ, only those candidate blocks that are indicated in C_φ need to be encoded. C_φ is downsampled to the so-called block picker C^B_φ, which has the same shape $(W/2, H/2)$ as $O^B_{z_0}$. Finally, the "Pick and Serialize" block in Fig. 5 picks the relevant elements from $O^B_{z_0}$ and from the output stack $\mathcal{G}_{z_0}$ and serializes them such that for each block $B_{m,n}$ for which $C^B_\varphi(m,n) = 1$, the 16-element pmf vector $\mathbf{G}(m,n)$ is extracted from the output stack, and the corresponding block occupancy symbol $O^B_{z_0}(m,n)$ is extracted from the block occupancy image. Then, the (symbol,pmf) pairs are sent to the encoder in a predefined scanning order.

Fig. 8 shows the decoder's flow diagram using the same probability modeling and the same CNN as the encoder. The decoder counterpart of the "Select and Serialize" block in the encoder is called the "Select (symbol,pmf) pairs" block. It performs perfectly aligned with the encoder. First,

it provides the arithmetic decoder with the pmf of the block to be decoded, and second, it reads from the output of the arithmetic decoder the decoded 2×2 block occupancies and inserts them into the correct locations in $R_{z_0}(\varphi)$. The essential difference from the way the input to the CNN is constructed at the encoder is the replacement of O_{z_0} by R_{z_0}.

D. OPTIMIZATION OF THE CNN

The CNN is trained using the data collected from a number of frames, which we call the training frames. The training set consists of $(4 \times 6 \times 6)$-shaped contexts that have occurred in the training frames at least once. Each context is associated with a 16-element histogram h containing the number of occurrences of 16 possible occupancy patterns of 2×2 candidate blocks. The training data are collected from the final resolution only.

During training, the contexts from the collected set are fed in batches of size N_b, and the loss is expressed as

$$Loss = -\frac{1}{N_b}\sum_{i=1}^{N_b}\sum_{q=0}^{15} h(q|\mathcal{C}_i)\log_2 g_q(\mathcal{W},\mathcal{C}_i), \tag{5}$$

where $h(q|\mathcal{C}_i)$ is the number of occurrences of the q'th occupancy pattern in the training set for the i'th context, $\mathcal{C}_i$, of the batch and $g_q(\mathcal{W},\mathcal{C}_i)$ is the corresponding output of the CNN; see (1).

III. EXPERIMENTAL WORK

We have performed experiments with sequences from Microsoft Voxelized Upper Bodies (MVUB) [33] and 8i Voxelized Full Bodies [34] datasets, which are dense point cloud datasets. MVUB sequences have a resolution of 9 bits, whereas 8i sequences have a resolution of 10 bits per dimension.

In our default CNN architecture (shown in Fig. 4), the total number of optimized parameters is 15116. Each parameter is transmitted in 4 bytes, leading to a model codelength of $\mathrm{CL}_m \tilde{=} 60.5$ kB. Thus, for a frame with 500k points in a 100-frame sequence, the model's contribution to the bitrate is less than 1%. Due to this small size, we did not consider, in this paper, the problem of entropy coding the CNN parameters, which will not significantly improve the currently achieved bitrates.

In all the experiments, training is performed using the ADAM optimizer with batch size $N_b = 10^4$ and an initial learning rate of 0.001. Since the training time is counted as part of the encoding time, the training phase is kept short. When no improvement in loss is observed for 20 epochs for the first time, the learning rate is halved. When no improvement in loss is observed once again, the training ends. The number of training frames is by default set to 5. To maximize parallelism, the sweeping axis Oz is selected as the shortest dimension of the bounding box, tightly enclosing all the points of the first frame. The algorithm is implemented using PyTorch and TorchAC [35] on an NVIDIA RTX 2080. The implementation is made available on Github.[1]

The bitrate br for the sequence is measured as the average of the bitrates br_f over F consecutive frames, where the bitrate br_f for a frame f is measured as bits-per-point (bpov)

$$\mathrm{br}_f = (\mathrm{CL}_f + \mathrm{CL}_m/F)/n_{p,f}, \tag{6}$$

and CL_f is the codelength for encoding frame f, CL_m is the codelength for encoding the CNN model and $n_{p,f}$ is the number of points in frame f.

In addition to the bitrates, we also report the average encoding and decoding times per frame, where the encoding time per frame t_e includes the time spent collecting training data and training the CNN divided by the number of frames. That is,

$$t_e = \frac{1}{F}(t_{tr} + \sum_{f=1}^{F} t_{e,f}), \tag{7}$$

where t_{tr} is the total time spent training the CNN (including the training data collection time) and $t_{e,f}$ is the time spent encoding frame f. For our default configuration, the average t_{tr} over all 9 sequences used in our experiments is 348 seconds.

In Table 1, we compare the bitrates br obtained with the proposed SeqNOC (default model) and with some recently published algorithms. The methods on the left side of the table have to resort to optimization (as in our method) or adaptively track the count numbers in each context to build efficient coding probability distributions. The right side of Table 1, under the header "Methods using models trained on a generic dataset" contains methods following the GTM paradigm [11], [17], [24], [26]. We note that for the *generically trained models*, there are no results listed for some sequences because some frames of those sequences were part of the training data; therefore, the cited publications did not include results for them. TMC13 [4], S3D [6], P(Full) [18], S4DCS [12] and SeqNOC results are the average bitrates over the 100 frames starting from the 2nd frame, whereas the results for NNOC, fNNOC [17] are for the entire sequence. VoxelDNN [11], MSVoxelDNN [26] and DGM [24] results are for certain representative frames taken from the sequences.

To investigate the effect of model complexity, we experiment with CNN models with different complexities. To that end, we devise three additional CNN model structures having the same number of convolutional layers as our default model, which is illustrated in Fig. 4 but with a different number of output channels at the hidden layers. These models are named SNLM, SNLM2 and SNHM (the first two of them being lighter than the default model and the last one being heavier), and their structures are summarized in Table 2, where n_w is the number of CNN parameters to be transmitted and $n_{hidden,l}$ is the number of output channels at the l'th hidden layer.

[1] https://github.com/marmus12/seqnoc

TABLE 1. Comparing the average bitrates of the proposed SeqNOC with those of the other recent methods.

	Methods using models adapting to the PC or to the sequence to be compressed						Methods using models trained on a generic dataset				
	TMC13 (v12.0) [5]	S3D [6]	P(Full) [18]	S4DCS [12]	FRL [32]	SeqNOC	fNNOC [17]	NNOC [17]	VDNN [11]	MSVDNN [26]	DGM [24]
Andrew	1.14	1.12	1.37	0.95	1.01	**0.85**	-	-	-	-	-
David	1.07	1.06	1.31	0.94	0.94	**0.78**	-	-	-	-	-
Phil	1.17	1.14	1.42	1.02	1.02	**0.86**	1.00	0.81	0.92	-	***0.76***
Ricardo	1.09	1.04	1.34	0.90	0.94	**0.79**	0.86	***0.68***	0.72	-	0.69
Sarah	1.07	1.07	1.37	0.92	0.96	**0.80**	-	-	-	-	-
Average$_{\text{MVUB}}$	1.11	1.09	1.36	0.95	0.97	**0.82**	-	-	-	-	-
LongDress	1.02	0.95	1.13	0.88	0.86	**0.70**	-	-	-	-	-
Loot	0.96	0.92	1.02	0.84	0.82	**0.66**	0.74	0.59	0.64	0.73	***0.58***
RedAndBlack	1.08	1.02	1.23	0.94	0.93	**0.77**	0.87	0.72	0.73	0.87	***0.66***
Soldier	1.03	0.96	0.85	**0.65**	0.88	0.70	-	-	-	-	-
Average$_{\text{8i}}$	1.02	0.96	1.06	0.83	0.87	**0.71**	-	-	-	-	-

TABLE 2. Structure of CNN models having different complexities.

	SeqNOC (default)	SNLM	SNLM2	SNHM
$n_{hidden,1}$	30	20	10	60
$n_{hidden,2}$	30	20	10	60
$n_{hidden,3}$	80	40	20	100
n_w	15116	5856	1736	41196

The bitrates and per-frame encoding-decoding times obtained with four different models are presented in Table 3, where the sequence length is $F = 100$. From Table 3, one can observe that the default model yields the best bpov results for the 9-bit MVUB sequences, whereas for the 10-bit 8i sequences, SNHM performs the best in bitrates. This is because, for the 9-bit sequences, the number of points at each point cloud is much less than the 10-bit point clouds so that the average bpov model cost of the heavy model (SNHM) is much more significant than the 10-bit case. A more complex model is more suitable when the number of points is high. On the other hand, the best decoding time is obtained with the lightest model (SNLM2); however, the bitrates of SNLM2 are the worst. Regarding the encoding time, there does not seem to be a clear-cut winner, but on average, SNLM performs the best. This is because the per-frame encoding time t_e is composed of both the time spent training the CNN t_{tr} (divided by the number of frames F) and the time spent encoding one frame. The time spent encoding one frame is shorter for the lighter models, whereas the CNN training time can sometimes be shorter when the model is more complex. Note that for all the experimental models, we employ the same training policy, which decides to end the training when the training loss stops to improve.

In Table 4, we compare the results obtained with different numbers of training frames. In our default configuration, the number of training frames is set to 5. Since the number of training frames does not affect the decoding times, we only present the encoding times in Table 4. According to Table 4, the per-frame encoding time t_e increases dramatically with the increasing number of training frames since the CNN training takes longer and the best encoding times are obtained in the single training frame case. For MVUB, the best bitrates are obtained with 20 training frames, whereas for the 8i dataset, the best bitrates are obtained with 10 training frames, yet the bitrate differences between the 10 and 20 training frame scenarios are rather small.

In Table 5, the results obtained with the default SeqNOC are compared with the single phase version of SeqNOC, which is called SeqNOC-SP. In SeqNOC-SP, we do not employ the four-phase strategy; instead, there is a single phase in which the probability of occupancies for a section $z = z_0$ is estimated. The SeqNOC-SP operates on less informative contexts, yet it has the advantage of speed. We perform this comparison mainly to show the improvement brought by the usage of phases.

Since the algorithms run at different speeds on different hardware, we compare the speed performance of different algorithms by comparing the ratios between the runtime of an algorithm and the runtime of TMC13 (reported in the publication describing the algorithm). The ratios between the average (per-frame) encoding/decoding times of the different methods and the average (per-frame) encoding/decoding times of TMC13 for PCs from MVUB and the 8i sequences are presented in Table 6. The encoding time ratio for FRL is taken from [32]. The runtimes for DGM, NNOC, fNNOC, MSVDNN, VoxelDNN and SeqNOC were measured on an NVIDIA RTX 2080, and the runtimes for TMC13 were measured as 2.9 seconds for encoding and 2.8 seconds for decoding on an Intel(R) Xeon(R) Silver 4110 CPU @ 2.10 GHz [24]. From Table 6, it is evident that FRL [32] is the fastest method, whereas SeqNOC-SP and SeqNOC are the third and fourth fastest, respectively. We note that the remaining methods are significantly slower than SeqNOC, and out of the NN-based methods, only the SeqNOC and SeqNOC-SP encoding times include the time spent for CNN training.

In Figure 9, we plot the average per-frame bitrate of several methods vs. the encoding time ratios to the TMC13 encoding time. Such 2D visualization allows us to roughly demonstrate the trade-off between bitrates and runtimes for different algorithms. In Fig. 9, the convex hull formed by

TABLE 3. Bitrates [bpov], encoding times (t_e[s]) and decoding times (t_d[s]) obtained with four different CNN models.

	SeqNOC (default)			SNLM			SNLM2			SNHM		
	bpov	t_e[s]	t_d[s]	bpov	t_e[s]	t_d[s]	bpov	t_e[s]	t_d[s]	bpov	t_e[s]	t_d[s]
Andrew	**0.85**	8.4	7.0	0.87	7.3	6.8	0.94	7.7	**6.6**	0.88	**7.1**	7.2
David	**0.78**	10.0	9.0	0.79	10.2	**8.5**	0.84	10.2	**8.5**	0.81	**9.3**	9.1
Phil	**0.86**	10.0	8.0	0.88	**9.1**	7.9	0.93	11.1	**7.3**	0.89	10.8	8.4
Ricardo	**0.79**	9.2	6.5	**0.79**	8.1	6.5	0.88	**5.8**	**6.2**	0.85	8.0	7.0
Sarah	**0.80**	9.2	8.5	0.82	**8.2**	7.9	0.87	10.0	**7.6**	0.85	9.1	8.2
Average$_{\text{MVUB}}$	**0.82**	9.4	7.8	0.83	**8.6**	7.5	0.89	9.0	**7.2**	0.86	8.9	8.0
Longdress	0.70	10.9	9.0	0.73	8.7	8.3	0.87	**7.9**	**8.1**	**0.68**	13.5	10.9
Loot	0.66	10.0	9.0	0.68	7.9	8.1	0.78	**7.4**	**7.9**	**0.65**	18.7	10.6
Redandblack	0.77	9.8	8.5	0.79	10.2	8.0	0.88	**8.7**	**7.5**	**0.76**	14.2	10.2
Soldier	0.70	12.7	10.0	0.74	**10.2**	9.3	0.8	11.8	**9.0**	**0.68**	15.8	12.1
Average$_{\text{8i}}$	0.71	10.8	9.1	0.73	9.3	8.4	0.83	**9.0**	**8.1**	**0.69**	15.6	11.0

TABLE 4. Bitrates [bpov] and encoding times (t_e[s]) for SeqNOC obtained with different numbers of training frames.

Number of Training Frames:	1		5 (default)		10		20	
	bpov	t_e[s]	bpov	t_e[s]	bpov	t_e[s]	bpov	t_e[s]
Andrew	0.91	**5.9**	0.85	8.4	**0.83**	9.5	**0.83**	10.0
David	0.84	**7.8**	0.78	10.0	**0.77**	11.9	**0.77**	11.2
Phil	0.93	**7.1**	0.86	10.0	0.84	12.5	**0.83**	21.0
Ricardo	0.86	**5.5**	0.79	9.2	0.77	11.8	**0.76**	16.5
Sarah	0.89	**6.5**	0.80	9.2	0.78	10.9	**0.77**	22.5
Average$_{\text{MVUB}}$	0.89	**6.6**	0.82	9.4	0.80	11.3	**0.79**	16.2
Longdress	0.72	**8.4**	0.70	10.9	**0.69**	14.2	0.70	16.7
Loot	0.67	**8.3**	0.66	10.0	**0.64**	16.4	0.65	17.5
Redandblack	0.79	**7.8**	0.77	9.8	**0.75**	16.0	0.76	16.9
Soldier	0.71	**11.3**	**0.70**	12.7	**0.70**	13.4	0.72	13.6
Average$_{\text{8i}}$	0.72	**9.0**	0.71	10.9	**0.70**	15.0	0.71	16.2

TABLE 5. Bitrates [bpov], encoding times (t_e[s]) and decoding times (t_d[s]) obtained with the default (4-phase) model vs. the single phase model.

	SeqNOC (default)			SeqNOC-SP		
	bpov	t_e[s]	t_d[s]	bpov	t_e[s]	t_d[s]
Andrew	**0.85**	8.4	7.0	0.90	**4.9**	**2.4**
David	**0.78**	10.0	9.0	**0.78**	**6.4**	**2.6**
Phil	**0.86**	10.0	8.0	0.88	**6.6**	**2.2**
Ricardo	**0.79**	9.2	6.5	0.80	**4.6**	**2.0**
Sarah	0.80	9.2	8.5	**0.79**	**5.9**	**2.2**
Average$_{\text{MVUB}}$	**0.82**	9.4	7.8	0.83	**5.7**	**2.3**
Longdress	**0.70**	10.9	9.0	0.75	**6.5**	**2.7**
Loot	**0.66**	10.0	9.0	0.70	**6.9**	**2.6**
Redandblack	**0.77**	9.8	8.5	0.81	**7.6**	**2.5**
Soldier	**0.70**	12.7	10.0	0.77	**6.4**	**3.0**
Average$_{\text{8i}}$	**0.71**	10.8	9.1	0.76	**6.9**	**2.7**

the datapoints corresponding to various algorithms is also drawn as a blue dashed line. The bitrates are averaged over 4 sequences, namely, phil, ricardo, loot and redandblack, for which the bitrates were available for all of the plotted methods. Similarly, in Fig. 10 are plotted the bitrates vs ratios of decoding times. From Figures 9 and 10, it is evident that both the default SeqNOC and its single-phase version SeqNOC-SP provide good trade-offs between encoding/decoding times and bitrates.

A. COMPRESSING A SINGLE POINT CLOUD WITH SeqNOC

Although the SeqNOC scheme is intended for the compression of sequences, it can also be used to compress a single point cloud, where the CNN optimization is performed with the context-histogram pairs extracted from the single point cloud. In the single point cloud case, the cost of transmitting the CNN parameters becomes quite significant; therefore, we choose to employ a CNN model with a small number of parameters. The model we use for the single point cloud case is SNLM2. The encoding time for a single PC is on the order of minutes since the time spent for optimization (t_{tr}) is on the order of minutes, whereas the decoding time is still on the order of seconds for a single point cloud. This property makes encoding of single frames by the STM strategy well suited for "encoding once decoding many times" scenarios (i.e., for broadcasting).

The bitrates obtained for the point clouds from Cat1A [36] are presented in Table 7. The point clouds with resolutions higher than 10 bits are downsampled to 10 bits. The bitrates reported in the SNLM2 column include the model cost (the cost of transmitting the model). Since the model costs for the single point cloud case are much more significant than those for the sequence case, the model costs for each PC are also shown separately in the Model Cost column. Note that for all of the PCs, the model structure and thus the model codelength $CL_m = 32 \times n_w$ are exactly the same, and the model cost in bpv (CL_m/n_p, where n_p is the number of points) varies due to the differences in the number of points in each PC. From Table 7, one can see that SNLM2 outperforms TMC13 for all the point clouds except three of them. Furthermore, SNLM2 offers significantly better bitrates on average when

TABLE 6. Ratios between the average (per-frame) runtime of different methods and the average (per-frame) runtime of TMC13.

	FRL [32]	SeqNOC-SP	SeqNOC (default)	fNNOC [17]	NNOC [17]	VoxelDNN [11]	MSVDNN [26]	DGM [24]
ENC	0.7	2.2	3.5	24.2	65.9	658.3	13.9	305.2
DEC	n.a.	2.5	3.0	27.5	418.2	2075.8	18.7	228.6

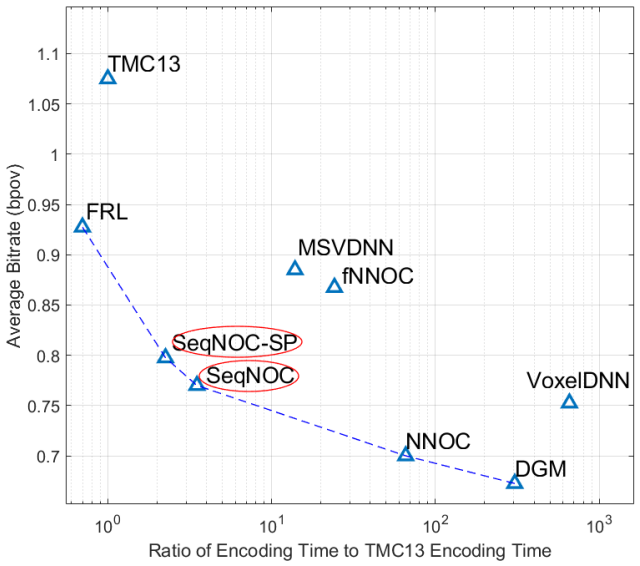

FIGURE 9. Average bitrate vs. encoding time ratio to TMC13 encoding time.

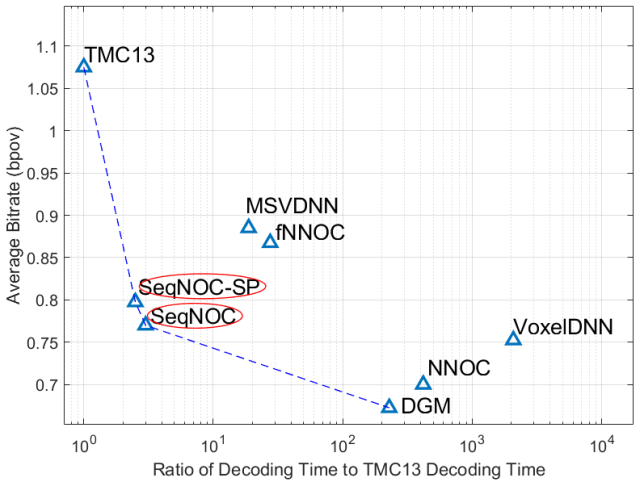

FIGURE 10. Average bitrate vs. decoding time ratio to TMC13 decoding time.

compared to TMC13. On the other hand, DGM [24] performs significantly better on shiva and frog.

IV. DISCUSSION

Our proposed scheme SeqNOC is a follow-up version of the schemes NNOC and fNNOC proposed in [17], with many architectural changes made with the main goal of achieving a reasonable per-frame *decoding time*. Additionally, from the encoder point of view, the structure of the neural network was changed and chosen to obtain a reasonable time for the optimal design of the CNN model based on several frames of the sequence. In a nutshell, we changed the schemes from [17] to allow a specific design of the CNN for each sequence with a *faster encoding time* (including even the time for optimizing a specific CNN for each sequence) and a faster decoding time, retaining almost as good lossless coding performance as in [17]. The similarities and differences are reviewed below in detail.

First, we discuss the similarities. The encoding of the occupancies of the voxels is performed in a multi-resolution fashion, sequentially encoding the octree representation at increasing resolutions. As with any context-based compression scheme, we tried to ensure the most informative contexts around the voxels to be encoded, using the voxels at the previous resolution (all of them being already encoded) and the voxels that are already encoded from the current resolution. In all SeqNOC and NNOC versions, we scan the voxels at a current resolution plane-by-plane (with each plane perpendicular on a chosen coordinate axis). Looking at the $4 \times L \times L$ ($L = 5$ for NNOC) neighbors of the voxel to be encoded in the current resolution, we see in the planes below it, $z = z_0 - 2$ and $z = z_0 - 1$, voxels for which the occupancy status is known, providing good contextual information. In the plane above, at $z = z_0 + 1$, the current resolution status is not known, but the previous resolution level pixels are already encoded, so we can create the candidate voxels at the current resolution (those that resulted from the split of an occupied voxel at the previous resolution). This is still useful contextual information but less useful than the information at $z = z_0 - 2$ and $z = z_0 - 1$.

In the current plane, $z = z_0$, one can use the voxels from the current resolution that have been already encoded (i.e., $\lfloor (L \times L - 1)/2 \rfloor$ voxels), as is done in NNOC, which results in a good coding performance, but this implies that the computation of the occupancy probabilities must be done at the decoder, one-by-one sequentially, for all the voxels in a plane $z = z_0$, resulting in very slow decoding. The fNNOC did not use all the voxels that have been already encoded from the current resolution at $z = z_0$, instead using only the candidate status at $z = z_0$, which resulted in faster decoding than NNOC, but with a lower compression performance. In SeqNOC, we use in a similar way the occupancy status of the neighbor voxels at $z = z_0 - 2$ and $z = z_0 - 1$ and the candidacy status of the voxels at $z = z_0 + 1$; however, the main change is in forming the context from the neighboring voxels at $z = z_0$.

TABLE 7. Bitrates [bpov] for TMC13, DGM, and for SNML2 (the lighter complexity version of SeqNOC, see Table 2) for the point clouds from Cat1A [36] in 10 bits resolution. The bitrate costs of transmitting the CNN parameters are included in the SNLM2 column but are also shown separately in the model cost column.

	TMC13	DGM [24]	SNLM2	Model Cost
shiva	3.70	**3.46**	3.58	0.06
loot	0.98	-	**0.81**	0.07
redandblack	1.11	-	**0.99**	0.07
longdress	1.03	-	**0.85**	0.06
queen	0.78	-	**0.70**	0.06
soldier	1.04	-	**0.85**	0.05
basketball player	0.87	-	**0.64**	0.07
head	1.54	-	**1.39**	0.02
boxer	0.96	-	**0.80**	0.06
facade9	2.41	-	**2.20**	0.06
facade15	1.58	-	**1.42**	0.07
facade64	1.03	-	**0.88**	0.05
dancer	0.86	-	**0.69**	0.08
frog	1.91	**1.70**	1.82	0.04
house without roof	**1.81**	-	2.00	0.03
thaidancer	**1.01**	-	1.02	0.20
arco valentino	**4.85**	4.99	4.86	0.04
Average	1.62	-	**1.50**	0.06

Some of the differences are listed next. To make a scheme even faster than fNNOC at the decoder and to improve the compression performance, in SeqNOC, we made two changes. First, we defined the element to be encoded at each arithmetic coding operation to be a block of 2×2 voxels (leading to a context of $4 \times 6 \times 6$ voxels around the 2×2 voxel block), instead of a single voxel in NNOC and fNNOC (for which the context had $4 \times 5 \times 5$ voxels). The prediction capabilities of the enhanced contexts in SeqNOC were shown in the ablation study to be better, not being affected too much by the dilution effect of the larger contexts. Another consequence of having a different symbol definition is the change in the shape of the context window.

The second change was to use a convolutional neural network instead of the multilayer neural network, which allowed us to compute all the probabilities at all the voxels very fast in parallel from a plane $z = z_0$. In NNOC, a two-layer MLP was employed, whereas in SeqNOC, a four-layer CNN, having fewer parameters than the MLP of NNOC, was employed. The CNN formulation enables faster execution, which helps to reduce the encoding/decoding times significantly.

Additionally, to further improve the compression performance, the pass of the encoding through plane $z = z_0$ was divided into four phases so that in each phase, the computation of the encoding probabilities can be performed in parallel but the contexts from the available voxels in $z = z_0$ are more informative, including the voxels $z = z_0$ already encoded in the previous phases.

The training stage is different. First, NNOC is a generically trained model, and therefore, the results obtained with NNOC are dependent on the training data. If the test data are not similar to the training data, the bitrate performance may deteriorate. In SeqNOC, the entropy model adapts to the input sequence itself. Since the symbols are defined in a different way than it was in NNOC, the loss function is also formulated in a different way.

In NNOC, during decoding, NN was executed for each candidate location in the current section; hence, the number of phases was as high as the number of candidates in the current section, resulting in very slow decoding. The four-phase approach in SeqNOC provides a good balance between bitrates and decoding speed.

Compared to the current state-of-the-art, our method is distinguished by two main features. First, the selection of the encoding unit (a 2×2 block of voxels) and its context (in the neighborhood of the block, utilizing the already encoded voxels at both the current resolution and the previous resolution of the octree), which allows encoding and decoding with rich contextual information in a parallel fashion along the plane-by-plane scanning of the point cloud at each resolution, which reaches competitive encoding and decoding speeds. Second, the introduction of lightweight CNN models, which can be trained quickly at the encoder and can be attached as a header to the bitstream for the full sequence without greatly affecting the overall bitrate, ensuring a very specific CNN model for the task at hand and alleviating the question of whether a generically trained CNN is suitable for the sequence at hand.

V. CONCLUSION

We have introduced a lossless geometry encoding scheme for sequences of dense point clouds using CNN models that are designed in a new paradigm, named *specifically trained models*, which has not been used until now. The learning of the CNN model can be done fast enough at the encoder, so that the learning plus sequence encoding time divided by the number of frames, gives very competitive per frame encoding times, at a bitrate performance better than that obtained in the competing paradigm of *generically trained models*. The coding probability models at each 2×2 block are computed in parallel by the convolutional neural network at each section $z = z_0$ through the point cloud, contributing to a fast encoding and decoding performance. To improve the decoding time, which has unreasonably large values for the published solutions [11], [17], [26], we adopt a four-phase encoding at each section, such that the content of the contexts improves from one phase to another, including the recently encoded/decoded voxels inside the context after each phase. Several variations of the method were proposed that covered various interesting trade-offs (e.g. compression ratio vs. time complexity). We discussed the performance of the proposed solution compared to the recently published schemes and found that the introduced features produce significant improvements.

REFERENCES

[1] S. Schwarz, M. Preda, V. Baroncini, M. Budagavi, P. Cesar, P. A. Chou, R. A. Cohen, M. Krivokuća, S. Lasserre, and Z. Li, "Emerging MPEG standards for point cloud compression," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 1, pp. 133–148, Mar. 2018.

[2] T. Ebrahimi, S. Foessel, F. Pereira, and P. Schelkens, "JPEG Pleno: Toward an efficient representation of visual reality," *IEEE Multimedia*, vol. 23, no. 4, pp. 14–20, Oct./Dec. 2016.

[3] E. S. Jang, M. Preda, K. Mammou, A. M. Tourapis, J. Kim, D. B. Graziosi, S. Rhyu, and M. Budagavi, "Video-based point-cloud-compression standard in MPEG: From evidence collection to committee draft [standards in a nutshell]," *IEEE Signal Process. Mag.*, vol. 36, no. 3, pp. 118–123, May 2019.

[4] D. Graziosi, O. Nakagami, S. Kuma, A. Zaghetto, T. Suzuki, and A. Tabatabai, "An overview of ongoing point cloud compression standardization activities: Video-based (V-PCC) and geometry-based (G-PCC)," *APSIPA Trans. Signal Inf. Process.*, vol. 9, no. 1, pp. 1–15, 2020.

[5] Moving Picture Experts Group. *TMC13*. Accessed: Mar. 20, 2020. [Online]. Available: https://github.com/MPEGGroup/mpeg-pcc-tmc13

[6] E. Peixoto, "Intra-frame compression of point cloud geometry using dyadic decomposition," *IEEE Signal Process Lett.*, vol. 27, pp. 246–250, 2020.

[7] E. Peixoto, E. Medeiros, and E. Ramalho, "Silhouette 4D: An inter-frame lossless geometry coder of dynamic voxelized point clouds," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2020, pp. 2691–2695.

[8] J. Wang, D. Ding, Z. Li, and Z. Ma, "Multiscale point cloud geometry compression," in *Proc. Data Compress. Conf. (DCC)*, Mar. 2021, pp. 73–82.

[9] H. Liu, H. Yuan, Q. Liu, J. Hou, and J. Liu, "A comprehensive study and comparison of core technologies for MPEG 3-D point cloud compression," *IEEE Trans. Broadcast.*, vol. 66, no. 3, pp. 701–717, Sep. 2020.

[10] S. Milani, E. Polo, and S. Limuti, "A transform coding strategy for dynamic point clouds," *IEEE Trans. Image Process.*, vol. 29, pp. 8213–8225, 2020.

[11] D. T. Nguyen, M. Quach, G. Valenzise, and P. Duhamel, "Learning-based lossless compression of 3D point cloud geometry," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Jun. 2021, pp. 4220–4224.

[12] E. Ramalho, E. Peixoto, and E. Medeiros, "Silhouette 4D with context selection: Lossless geometry compression of dynamic point clouds," *IEEE Signal Process. Lett.*, vol. 28, pp. 1660–1664, 2021.

[13] E. C. Kaya, S. Schwarz, and I. Tabus, "Refining the bounding volumes for lossless compression of voxelized point clouds geometry," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2021, pp. 3408–3412.

[14] M. Quach, G. Valenzise, and F. Dufaux, "Learning convolutional transforms for lossy point cloud geometry compression," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2019, pp. 4320–4324.

[15] D. C. Garcia and R. L. de Queiroz, "Context-based octree coding for point-cloud video," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2017, pp. 1412–1416.

[16] L. Huang, S. Wang, K. Wong, J. Liu, and R. Urtasun, "OctSqueeze: Octree-structured entropy model for LiDAR compression," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 1310–1320.

[17] E. C. Kaya and I. Tabus, "Neural network modeling of probabilities for coding the octree representation of point clouds," in *Proc. IEEE 23rd Int. Workshop Multimedia Signal Process. (MMSP)*, Oct. 2021, pp. 1–6.

[18] D. C. Garcia, T. A. Fonseca, R. U. Ferreira, and R. L. de Queiroz, "Geometry coding for dynamic voxelized point clouds using octrees and multiple contexts," *IEEE Trans. Image Process.*, vol. 29, pp. 313–322, 2019.

[19] R. L. de Queiroz, D. C. Garcia, P. A. Chou, and D. A. Florencio, "Distance-based probability model for octree coding," *IEEE Signal Process. Lett.*, vol. 25, no. 6, pp. 739–742, Jun. 2018.

[20] Z. Que, G. Lu, and D. Xu, "VoxelContext-Net: An octree based framework for point cloud compression," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 6042–6051.

[21] J. Wang, H. Zhu, H. Liu, and Z. Ma, "Lossy point cloud geometry compression via end-to-end learning," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 12, pp. 4909–4923, Dec. 2021.

[22] X. Wen, X. Wang, J. Hou, L. Ma, Y. Zhou, and J. Jiang, "Lossy geometry compression of 3D point cloud data via an adaptive octree-guided network," in *Proc. IEEE Int. Conf. Multimedia Expo (ICME)*, Jul. 2020, pp. 1–6.

[23] D. E. O. Tzamarias, K. Chow, I. Blanes, and J. Serra-Sagristà, "Compression of point cloud geometry through a single projection," in *Proc. Data Compress. Conf. (DCC)*, Mar. 2021, pp. 63–72.

[24] D. T. Nguyen, M. Quach, G. Valenzise, and P. Duhamel, "Lossless coding of point cloud geometry using a deep generative model," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 12, pp. 4617–4629, Dec. 2021.

[25] M. Quach, J. Pang, D. Tian, G. Valenzise, and F. Dufaux, "Survey on deep learning-based point cloud compression," *Frontiers Signal Process.*, vol. 2, pp. 1–15, Feb. 2022.

[26] D. T. Nguyen, M. Quach, G. Valenzise, and P. Duhamel, "Multiscale deep context modeling for lossless point cloud geometry compression," in *Proc. IEEE Int. Conf. Multimedia Expo Workshops (ICMEW)*, Jul. 2021, pp. 1–6.

[27] A. F. R. Guarda, N. M. M. Rodrigues, and F. Pereira, "Adaptive deep learning-based point cloud geometry coding," *IEEE J. Sel. Topics Signal Process.*, vol. 15, no. 2, pp. 415–430, Feb. 2021.

[28] D. Meagher, "Geometric modeling using octree encoding," *Comput. Graph. Image Process.*, vol. 19, no. 2, pp. 129–147, Jun. 1982.

[29] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, 2013, vol. 30, p. 3.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1026–1034.

[31] M. Alzantot, Z. Wang, and M. B. Srivastava, "Deep residual neural networks for audio spoofing detection," 2019, *arXiv:1907.00501*.

[32] D. E. O. Tzamarias, K. Chow, I. Blanes, and J. Serra-Sagrista, "Fast run-length compression of point cloud geometry," *IEEE Trans. Image Process.*, vol. 31, pp. 4490–4501, 2022.

[33] C. Loop, Q. Cai, S. O. Escolano, and P. A. Chou, *Microsoft Voxelized Upper Bodies—A Voxelized Point Cloud Dataset*, document ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) Input document m38673/M72012, 2016.

[34] E. d'Eon, B. Harrison, T. Myers, and P. A. Chou, *8I Voxelized Full Bodies—A Voxelized Point Cloud Dataset*, document ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) Input document WG11M40059/WG1M74006, 2017.

[35] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. Van Gool, "Practical full resolution learned lossless image compression," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 10629–10636.

[36] S. Schwarz, G. Martin-Cocher, D. Flynn, and M. Budagavi, *Common Test Conditions for Point Cloud Compression*, document ISO/IEC JTC1/SC29/WG11 w17766, Ljubljana, Slovenia, 2018.

[37] R. Mekuria, K. Blom, and P. Cesar, "Design, implementation, and evaluation of a point cloud codec for tele-immersive video," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 27, no. 4, pp. 828–842, Apr. 2017.

[38] C. Cao, M. Preda, V. Zakharchenko, E. S. Jang, and T. Zaharia, "Compression of sparse and dense dynamic point clouds—Methods and standards," *Proc. IEEE*, vol. 109, no. 9, pp. 1537–1558, Sep. 2021.

[39] R. L. de Queiroz and P. A. Chou, "Motion-compensated compression of dynamic voxelized point clouds," *IEEE Trans. Image Process.*, vol. 26, no. 8, pp. 3886–3895, Aug. 2017.

[40] D. C. Garcia and R. L. de Queiroz, "Intra-frame context-based octree coding for point-cloud geometry," in *Proc. 25th IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2018, pp. 1807–1811.

[41] S. Milani, "Fast point cloud compression via reversible cellular automata block transform," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2017, pp. 4013–4017.

[42] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach, "Real-time compression of point cloud streams," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2012, pp. 778–785.

[43] D. Lazzarotto, E. Alexiou, and T. Ebrahimi, "On block prediction for learning-based point cloud compression," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2021, pp. 3378–3382.

[44] T. Huang and Y. Liu, "3D point cloud geometry compression on deep learning," in *Proc. 27th ACM Int. Conf. Multimedia*, Oct. 2019, pp. 890–898.

[45] L. Gao, T. Fan, J. Wan, Y. Xu, J. Sun, and Z. Ma, "Point cloud geometry compression via neural graph sampling," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2021, pp. 3373–3377.

[46] H. Roodaki, M. Dehyadegari, and M. N. Bojnordi, "G-Arrays: Geometric arrays for efficient point cloud processing," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Jun. 2021, pp. 1925–1929.

[47] Y. Xu, W. Zhu, Y. Xu, and Z. Li, "Dynamic point cloud geometry compression via patch-wise polynomial fitting," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2019, pp. 2287–2291.

EMRE C. KAYA was born in Edirne, Türkiye, in 1990. He received the B.S. and M.S. degrees from the Department of Electrical and Electronics Engineering, Middle East Technical University, Ankara, Türkiye, in 2015 and 2018, respectively. He is currently pursuing the Ph.D. degree with the Computing Sciences Unit, Tampere University, under the supervision of Prof. I. Tabus. His research interests include point cloud compression, image compression, and visual object detection.

IOAN TABUS (Senior Member, IEEE) received the Ph.D. degree (Hons.) from the Tampere University of Technology, Finland, in 1995.

He held teaching positions with the Department of Control and Computers, Politehnica University of Bucharest, from 1984 to 1995. Since 1996, he has been a Senior Researcher, and since January 2000, he has been a Professor with the Department of Signal Processing, Tampere University of Technology, which was merged into Tampere University, in 2019. He is the coauthor of two books and more than 250 publications in the fields of signal compression, image processing, bioinformatics, and system identification. His research interests include light field image processing, plenoptic image compression, point cloud compression, audio, image, data compression, genomic signal processing, and statistical signal processing. He was a co-recipient of the 1991 Train Vuia Award of Romania, the 2001 NSIP Best Paper Award, the 2004 NORSIG Best Paper Award, the 2016 3DTV Best Paper Award, and the ICIP 2017 Light Field Image Coding Challenge Award. He is an Associate Editor of the IEEE TRANSACTIONS ON IMAGE PROCESSING. He served as an Associate Editor for the IEEE TRANSACTIONS ON SIGNAL PROCESSING and *EURASIP Journal on Advances in Signal Processing*. He has served as a Guest Editor for special issues for the *IEEE Signal Processing Magazine*, *EURASIP Journal on Advances in Signal Processing*, and the IEEE JOURNAL OF SELECTED TOPICS IN SIGNAL PROCESSING. He was the Editor-in-Chief of the *EURASIP Journal on Bioinformatics and Systems Biology*, from 2006 to 2014.