Deep-PCAC: An End-to-End Deep Lossy Compression Framework for Point Cloud Attributes

摘要

近年来，随着点云技术在计算机视觉、机器学习和虚拟现实等领域的广泛应用，其在表示和捕捉三维物体及场景信息方面的优势逐渐受到了重视。然而，点云数据的庞大规模和高维特征使得在存储和传输方面面临着严峻的挑战。为解决这一问题，本文引入了一项创新性的工作，提出了首个端到端的深度学习点云属性压缩方案，旨在实现对点云属性的高效有损压缩。在复现工作中，我们着重实现了压缩方案的编码器部分及相应的训练代码。通过在原始数据集上进行充分的测试，我们验证了该方案在不同场景和物体上的适用性和稳健性。实现的编码器部分不仅具备高效的压缩性能，同时在保留点云属性重要信息的同时，显著减小了存储和传输所需的数据量。

关键词：点云；点云属性压缩；熵模型

1 引言

点云数据作为一种重要的三维表示形式，广泛应用于计算机视觉、机器学习以及虚拟现实等领域。在处理大规模、高分辨率的点云数据时，传输和存储成本显著增加，因此点云压缩成为一项关键任务。本文聚焦于点云属性的压缩，旨在提出一种高效、端到端的深度点云有损属性压缩方法，以克服大规模点云数据传输和存储方面的挑战。

在这篇论文中，作者首先引入了深度点云有损属性压缩（Deep-PCAC）方法，该方法通过端到端的方式直接对点云属性进行编解码。与传统方法不同，作者采用集合学习的思想，充分利用点云中点之间的关系，以提高压缩效率。这使得该方法能够更全面地捕捉点云的特征，从而更准确地还原实际物体的表面或整体场景。

接着，在编码器的设计中，作者引入了二阶点卷积，通过更好地建模多点之间的复杂关系，进一步提高了属性压缩的性能。这种创新性的设计在保留点云细节的同时，有效减少了压缩后的数据量，为大规模点云的传输和存储提供了更为经济的解决方案。

此外，作者还引入了多尺度损失，以引导编码器更加专注于粗粒度的点。这种损失机制有助于更好地覆盖整个点云，使得压缩后的数据在保持高质量细节的同时，能够更加紧凑地表示整体形状特征。这一创新性的多尺度损失策略在提高点云属性压缩效率的同时，保证了对整体结构的准确还原。

2 相关工作

2.1 点云属性压缩

在近年来，针对点云属性压缩问题，研究者们提出了众多基于变换的方法。其中，图傅里叶变换作为一种广泛应用于去相关非结构化信号的工具，在点云属性压缩领域引起了重要关注。这种图变换充分利用了点云局部邻域的空间相关性，相对于直接对属性进行 DCT 等变换，表现出更卓越的压缩性能 [?]"03032014$_I CIP_Z hang_P oint$) $Queiroz$ GPs

2.2 基于学习的点云压缩

近年来，随着深度学习的迅猛发展，基于学习的点云压缩方案逐渐成为研究的热点，然而，大多数研究主要集中在点云的几何压缩方面。在这方面，Quach 等人 [1]、Wang 等人 [2] 以及 Guarda[3] 等学者通过采用基于 3D CNN 的自编码器架构，尝试对点云的 3D 体素化模型进行表示，并通过二分类损失函数（BCE 损失）进行优化，将占用重建问题转化为分类问题。这种方法不仅能够有效地捕捉点云的几何特征，而且为点云的深度学习压缩提供了有益的思路。另一方面，Wang 等人 [4] 提出了一种基于稀疏卷积的点云几何压缩模型。通过考虑点云的稀疏性，他们在时间和空间维度上降低了计算复杂度，从而在保留关键信息的同时提高了压缩效率。这种稀疏卷积的引入为点云几何压缩领域带来了新的可能性，有效地解决了传统方法在处理大规模点云时的计算困难。另外一些研究者专注于基于点的压缩模型。例如，Huang 等人 [5] 设计了一种层次化模型，具有多尺度损失，用于细粒度的点云重建。这种层次化模型通过引入多尺度损失，能够更全面地捕捉点云中的特征，从而实现更高质量的细粒度重建。另一项有趣的工作由 Wen 等人 [6] 提出，他们设计了一种自适应的八叉树引导网络，用于大规模点云的重建。这种方法通过引入自适应的八叉树结构，能够在处理大规模点云时更好地平衡计算复杂度和重建质量，为大规模点云场景的深度学习压缩提供了一种有效的解决方案。

3 本文方法

3.1 本文方法概述

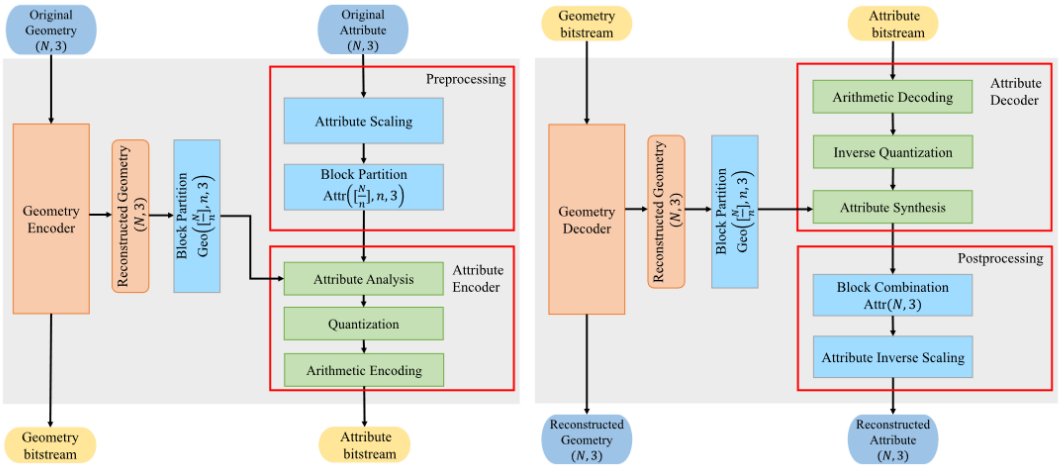

图 1. 方法示意图

本研究的方法，如图 1所示，主要包括三个关键组成部分：点云属性自编码器、预处理模块以及后处理模块。点云属性自编码器通过充分利用几何结构对点云属性进行编码和解码。为了更有效地进行条件概率建模，作者在自编码器中引入了超先验模型。属性编码器生成一个紧凑的潜在代码，而属性解码器则对称地重构原始属性，形成一个完整的编解码过程。此外，作者提出了一种专注于属性缩放和块划分的预处理模块，以及一个后处理模块，用于对块进行重新组合和进行属性的逆缩放操作。这一系列的组件协同工作，构成了深度学习损失压缩框架的核心，以期在点云数据领域取得显著的性能提升。

3.2 编解码器

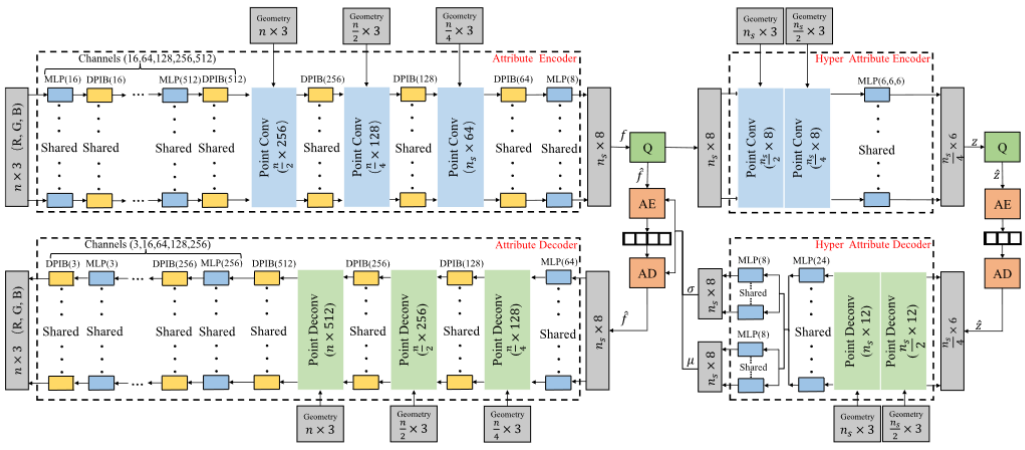

图 2. 编解码器

图 2展示了本研究的编解码器结构，其中包括五个关键组件：属性编码器，属性解码器，超先验模型，熵模型以及损失函数。具体而言，属性编码器的任务是将原始点云属性转换为潜在特征编码。该过程首先利用多层感知器（MLP）和密集点云感知块（DBIP）的组合提取属性特征，随后通过二阶点卷积层进行特征聚合。在 DBIP 中，采用双分支结构以减少计算复杂性并促进信息传播。对于具有 C 个通道的输入，每个分支仅计算 $\frac{C}{2}$ 个通道，而二阶点卷积层能够利用更多点之间的空间相关性。属性解码器则是编码器的逆操作，通过隐向量还原为原始属性。超先验模型用于缩小量化后的潜在编码，从而提供超先验信息。超属性解码器通过解码后的超先验获得潜在编码中每个值的估计均值 μ 和方差 θ。这两个参数 μ 和 θ 被用作先验知识，用于优化潜在属性特征的概率估计，进一步提高模型性能。

3.3 预处理与后处理模块

预处理模块的目标是将包含数百万个点的点云划分成小块，以便将每个块输入属性自编码器进行压缩。该模块包括两个主要步骤：属性缩放和块划分。在属性缩放步骤中，点云属性被缩放到标准化的范围 [01]。而在块划分步骤中，整个点云，其点数可能为 N，被分割成具有固定点数 n 的局部块。

相对应于预处理模块，后处理模块的任务是将解码后的块组织成完整的点云。后处理模块同样包括两个关键步骤：块组合和属性反比例缩放。块组合阶段负责将解码后的块重新组合，以还原点云的完整结构。而属性反比例缩放阶段通过乘以属性缩放模块中使用的因子 T，将属性值从标准化范围 [01] 映射回原始范围 [0255]。

3.4 损失函数定义

当采用常见的率失真优化损失函数优化自编码器时，公式可以分为几个主要部分：

$$J_{loss} = R + \lambda \cdot D$$

其中，R 表示比特率，由属性隐码 $R_{\hat{f}}$ 和超先验 $R_{\hat{z}}$ 的比特流组成。λ 是用于平衡比特率和失真的权重参数。

$$R = R_{\hat{f}} + R_{\hat{z}}$$

D 是逐点重建损失和多尺度损失的加权和：

$$D = D_{RPL} + \alpha \cdot D_{MSL}$$

其中，D_{RPL} 是原始点云和解码点云之间每个点的属性误差，在计算之前需要将点云的颜色属性转换到 YUV 空间中：而 D_{MSL} 是多尺度损失。逐点重建损失衡量了原始点云和解码点云之间每个点的属性误差，其中 y_i 表示原始点云中第 i 个点的属性。多尺度损失通过逐点属性误差 D_{PRL} 对多个尺度进行加权求和，其中 $Y_{i,j}$ 表示原始点云中第 i 个点在第 j 个尺度上的属性，而 $\hat{Y}_{i,j}$ 表示解码点云上对应的属性。多尺度的引入有助于更全面地捕获点云中的特征，提高压缩模型的性能。

$$D_{RPL}(Y, \hat{Y}) = \sum_{i=1}^{n} \|y_i - \hat{y}_i\|_2^2$$

$$D_{MSL}(Y,\hat{Y}) = \sum_{j}^{M}\sum_{i}^{N_j} D_{PRL}(Y_{i,j},\hat{Y}_{i,j})$$

其中，y_i 表示原始点云中第 i 个点的属性。

4 复现细节

4.1 与已有开源代码对比

开源代码只提供了部分测试代码，我按照论文内容实现了模型训练部分的代码以及编码器部分的代码，包括属性编码器，属性解码器，超先验模型以及损失函数。

4.2 代码分析

4.2.1 编码器

```
class AttributeEncoder(nn.Module):
    def __init__(self,Nin,Ns):
        super(AttributeEncoder, self).__init__()
        self.mlp1 = MLP(inputchannel=3, outputchannel=16)
        self.mlp2 = MLP(inputchannel=16, outputchannel=64)
        self.mlp3 = MLP(inputchannel=64, outputchannel=128)
        self.mlp4 = MLP(inputchannel=128, outputchannel=256)
        self.mlp5 = MLP(inputchannel=256, outputchannel=512)
        self.dpib1 = DPIB(channel=16)
        self.dpib2 = DPIB(channel=64)
        self.dpib3 = DPIB(channel=128)
        self.dpib4 = DPIB(channel=256)
        self.dpib5 = DPIB(channel=512)

        self.PointConv1=PointConv(Nout=int(Nin/2),
                                  featurein=512,
                                  featureout=256)
        self.dpib6 = DPIB(channel=256)

        self.PointConv2 = PointConv(Nout=int(Nin / 4),
                                    featurein=256,
                                    featureout=128)
        self.dpib7 = DPIB(channel=128)

        self.PointConv3 = PointConv(Nout=Ns,
                                    featurein=128,
```

```
                                        featureout=64)
        self.dpib8 = DPIB(channel=64)

        self.mlp6 = MLP(inputchannel=64,
                        outputchannel=8)


    def forward(self,rgb,geometry):
        x = self.dpib1(self.mlp1(rgb))
        x = self.dpib2(self.mlp2(x))
        x = self.dpib3(self.mlp3(x))
        x = self.dpib4(self.mlp4(x))
        x = self.dpib5(self.mlp5(x))
        x,geometry1 = self.PointConv1(x,geometry)
        x = self.dpib6(x)
        x, geometry2 = self.PointConv2(x, geometry1)
        x = self.dpib7(x)
        x, geometry3 = self.PointConv3(x, geometry2)
        x = self.dpib8(x)
        x=self.mlp6(x)
        return x,geometry3
```

上述代码实现了编码器部分，用于点云属性的端到端编码。通过多层感知机（MLP）和密集点云感知块（DPIB）的组合，以及使用点云卷积层（PointConv）进行属性的处理和降维，该模型能够有效地提取输入点云的深层特征。最终输出具有 8 个通道的编码后的点云属性。

4.2.2 解码器

```
class AttributeDecoder(nn.Module):
    def __init__(self,Nout,Ns):
        super(AttributeDecoder, self).__init__()
        self.Nout=Nout
        self.Ns=Ns
        self.mlp1 = MLP(inputchannel=8, outputchannel=64)

        self.PointDec1=PointDeconv(Nout=int(Nout/4),
                                   featurein=64,
                                   featureout=128)
        self.dpib1=DPIB(channel=128)
        self.PointDec2 = PointDeconv(Nout=int(Nout / 2),
                                     featurein=128,
```

```
                                        featureout=256)
        self.dpib2=DPIB(channel=256)
        self.PointDec3 = PointDeconv(Nout=int(Nout / 1),
                                        featurein=256,
                                        featureout=512)
        self.dpib3 = DPIB(channel=512)

        self.mlp4 = MLP(inputchannel=512, outputchannel=256)
        self.dpib4 = DPIB(channel=256)
        self.mlp5 = MLP(inputchannel=256, outputchannel=128)
        self.dpib5 = DPIB(channel=128)
        self.mlp6 = MLP(inputchannel=128, outputchannel=64)
        self.dpib6 = DPIB(channel=64)
        self.mlp7 = MLP(inputchannel=64, outputchannel=16)
        self.dpib7 = DPIB(channel=16)
        self.mlp8 = MLP(inputchannel=16, outputchannel=3)
        self.dpib8 = DPIB(channel=3)

    def forward(self,latentcode,geometry):
        geolist=GeometrySample(geometry,N=self.Nout,Ns=self.Ns)

        x = self.mlp1(latentcode)
        x = self.dpib1(self.PointDec1(x, geolist[3], geolist[2]))
        x = self.dpib2(self.PointDec2(x, geolist[2], geolist[1]))
        x = self.dpib3(self.PointDec3(x, geolist[1], geolist[0]))

        x = self.dpib4(self.mlp4(x))
        x = self.dpib5(self.mlp5(x))
        x = self.dpib6(self.mlp6(x))
        x = self.dpib7(self.mlp7(x))
        x = self.dpib8(self.mlp8(x))

        return x
```

上述代码实现了解码器部分，用于点云属性的解码。模型由一系列 MLP 层、PointDeconv 层和密集点云感知块（DPIB）层组成，构建了一个端到端的解码网络。在解码过程中，通过逐渐减小通道数和使用反卷积层，模型还原了点云属性的空间结构。值得注意的是，为了更好地适应点云的特性，模型在解码过程中引入了与输入几何信息相关的 `GeometrySample` 函数。这一结构设计有助于提高解码的精度和对原始点云的准确重建。整体而言，`AttributeDecoder` 的结构使其成为一个有效的端到端点云属性解码器，为点云处理任务提供了有力的工具。

4.2.3 超先验模型

```
class HyperEncoder(nn.Module):
    def __init__(self,Ns):
        super(HyperEncoder, self).__init__()
        self.PointConv1=PointConv(Nout=int(Ns/2),
                                  featurein=8,
                                  featureout=8)
        self.PointConv2 = PointConv(Nout=int(Ns / 4),
                                    featurein=8,
                                    featureout=8)
        self.mlp1=MLP(inputchannel=8,outputchannel=6)
        self.mlp2 = MLP(inputchannel=6, outputchannel=6)
        self.mlp3 = MLP(inputchannel=6, outputchannel=6)
    def forward(self,x,geoin):

        x,geo=self.PointConv1(x,geoin)
        x,geo_=self.PointConv2(x,geo)
        x=self.mlp1(x)
        x = self.mlp2(x)
        x = self.mlp3(x)
        return x,geo_

class HyperDecoder(nn.Module):
    def __init__(self,Ns):
        super(HyperDecoder, self).__init__()
        self.Ns=Ns
        self.PointDec1=PointDeconv(Nout=int(Ns/2),
                                   featurein=6,
                                   featureout=12)
        self.PointDec2 = PointDeconv(Nout=Ns,
                                     featurein=12,
                                     featureout=12)
        self.mlp=MLP(inputchannel=12,outputchannel=24)
        self.mlp1 = MLP(inputchannel=24, outputchannel=8)
        self.mlp2 = MLP(inputchannel=24, outputchannel=8)
    def forward(self,x,geo):
        ls=GeometrySample(geometry=geo,N=self.Ns,Ns=self.Ns)
        x=self.PointDec1(x,ls[2],ls[1])
        x=self.PointDec2(x,ls[1],ls[0])
        x=self.mlp(x)
```

```
        ave=self.mlp1(x)
        std=self.mlp2(x)
        return ave,std
```

4.2.4 训练部分

```
if __name__ == "__main__":
    args = parse_args()
    if args.gpu==1:
        os.environ['CUDA_VISIBLE_DEVICES']="0"
    else:
        os.environ['CUDA_VISIBLE_DEVICES']=""
    config = tf.ConfigProto()
    config.gpu_options.per_process_gpu_memory_fraction = 1.0
    config.gpu_options.allow_growth = True
    config.log_device_placement=True
    sess = tf.Session(config=config)


    # Define your training hyperparameters
    epochs = 100
    batch_size = 32
    learning_rate = 0.001

    optimizer = tf.keras.optimizers.Adam(
                learning_rate=learning_rate)

    # Training loop
    for epoch in range(epochs):
        rootdir, filename = os.path.split(args.input)
        with tf.GradientTape() as tape:
            # Forward pass
            point_set,num_less_than_2048 = preprocess(args.input)
            x_coori = point_set[:,:,0:3]
            x_color = point_set[:,:,3:6]
            y_strings, y_min_v, y_max_v, y_shape,
            z_strings, z_min_v, z_max_v, z_shape =
            compress(x_coori,x_color, args.model,
            args.ckpt_dir,args.latent_points)
            args.output = filename + "_rec.ply"
```

```
            ori_cooridinate_path = args.input + ".ply"
            y_strings_d, z_strings_d, num_less_than_2048_d, \
            y_min_v_d, y_max_v_d, y_shape_d,
            z_min_v_d, z_max_v_d, z_shape_d =
            read_binary_files(filename, './compressed')

            point_set_ori,num_less_than_2048 =
            preprocess(ori_cooridinate_path)
            ori_coori = point_set_ori[:,:,0:3]

            rec_color = decompress
            (ori_coori,y_strings_d, y_min_v_d, y_max_v_d, y_shape_d,
            z_strings_d, z_min_v_d, z_max_v_d, z_shape_d, args.model,
            args.ckpt_dir,args.latent_points)
            ori_coori = point_set_ori[:,:,0:3]
            rec_point_cloud = np.concatenate(
            (ori_coori,rec_color),-1)
            pc_rec = postprocess(args.output,
            rec_point_cloud, int(num_less_than_2048_d),
            points_num=2048)

            loss = psnr(point_set, pc_rec)

        # Print or log the training loss at the end of each epoch
        print(f"Epoch {epoch + 1}/{epochs}, Loss: {loss.numpy()}")
        # Backpropagation
        gradients = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(gradients,
        model.trainable_variables))
```

5 实验结果分析

在实验部分，采用原文中使用的 MPEG PCCv2[7] 点云数据集，其中包括 RedandBlack，Queen，Longdress，Loot。将点云分割成 2048 块。我们在三个不同的数据集上进行了测试。第一个数据集 [7] 包括' Mask '，' Frog '，' Staue Klimt '，' Shiva '，' Soldier ' 和' House '，它们是 PCC 第 1 类通用测试条件中定义的体素化点云。第二个数据集 [8] 包括斯坦福三维大规模室内空间数据集的两个非体素化区域，即" Area2 " 和" Area4 "。第三个数据集包含来自 Sketchfab 网站的 4 个未体素化的点云，命名为' Explode Hair '，' Goldberg '，' Vase ' 和' Woman '。在实验中，这些数据集被用于测试模型的性能，并评估压缩结果在点云重建方面的质量。这些数据集的选择涵盖了不同类型的点云和不同场景的特征，有助于全面评估模型在

各种情况下的性能。

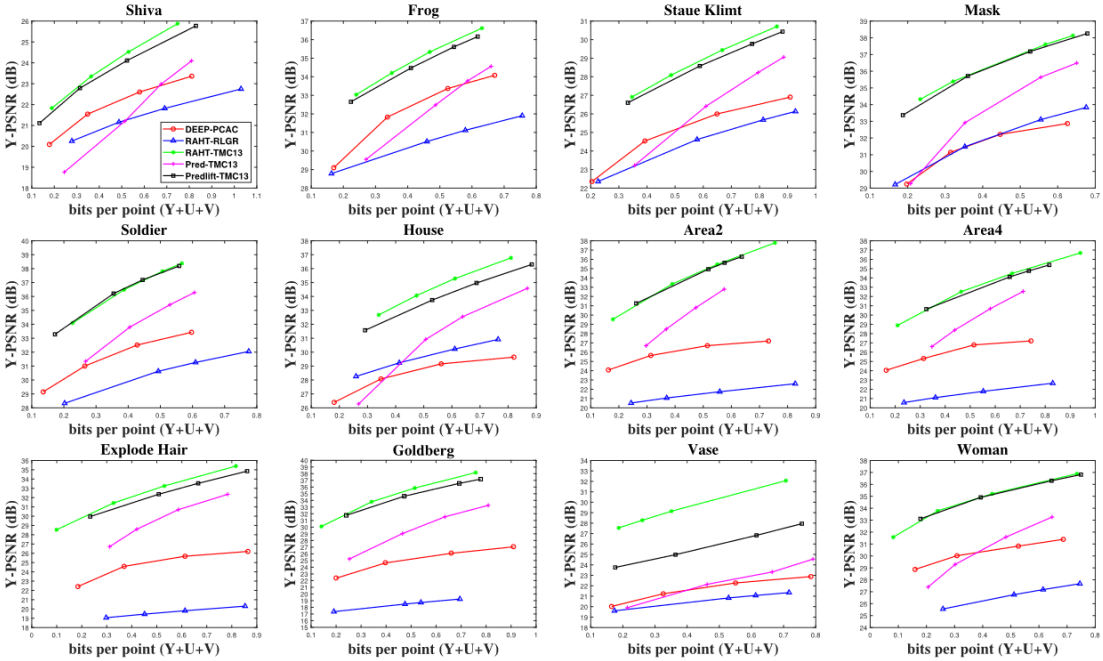

图 3. 实验结果

在图 3中展示了实验结果。观察到，与当前最先进的方法相比，本文提出的端到端深度学习压缩框架在性能上存在一定的损失。与 RAHT-TM13 和 Predlift-TMC13 相比，本文的方法分别具有 5.67 dB 和 5.29 dB 的性能差距。

尽管存在一定的性能差距，但本文提出的方法仍具有一定的价值。作为第一个端到端的基于学习的点云属性压缩框架，这项工作为未来的点云属性压缩研究提供了新的方向。实验结果不仅反映了模型的性能，还为后续工作提供了改进和优化的空间。可能需要进一步的实验和分析，以更全面地理解模型性能的局限性，并探索提高性能的潜在改进方向。

6 总结与展望

在总结中，本文提出了一种端到端的深度学习点云属性压缩框架，通过点云属性自编码器、预处理模块和后处理模块的组合，实现了对点云的有效压缩。实验结果表明，尽管与当前最先进的方法相比存在一定性能差距，但这一框架作为首个基于学习的点云属性压缩方案，为未来的研究提供了新的方向。展望未来，我们可以通过进一步优化模型结构、引入更复杂的超先验模型以及探索更先进的损失函数，提高压缩性能。此外，拓展实验数据集的多样性和规模，以更全面地评估模型在不同场景下的性能，将是未来工作的关键方向。

References

[1] Maurice Quach, Giuseppe Valenzise, and Frederic Dufaux. Learning convolutional transforms for lossy point cloud geometry compression. In *2019 IEEE international conference on image processing (ICIP)*, pages 4320–4324. IEEE, 2019.

[2] Jianqiang Wang, Hao Zhu, Zhan Ma, Tong Chen, Haojie Liu, and Qiu Shen. Learned point cloud geometry compression. *arXiv preprint arXiv:1909.12037*, 2019.

[3] André FR Guarda, Nuno MM Rodrigues, and Fernando Pereira. Point cloud coding: Adopting a deep learning-based approach. In *2019 Picture Coding Symposium (PCS)*, pages 1–5. IEEE, 2019.

[4] Jianqiang Wang, Dandan Ding, Zhu Li, and Zhan Ma. Multiscale point cloud geometry compression. In *2021 Data Compression Conference (DCC)*, pages 73–82. IEEE, 2021.

[5] Tianxin Huang and Yong Liu. 3d point cloud geometry compression on deep learning. In *Proceedings of the 27th ACM international conference on multimedia*, pages 890–898, 2019.

[6] Xuanzheng Wen, Xu Wang, Junhui Hou, Lin Ma, Yu Zhou, and Jianmin Jiang. Lossy geometry compression of 3d point cloud data via an adaptive octree-guided network. In *2020 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2020.

[7] Sebastian Schwarz, Gaëlle Martin-Cocher, David Flynn, and Madhukar Budagavi. Common test conditions for point cloud compression. *Document ISO/IEC JTC1/SC29/WG11 w17766, Ljubljana, Slovenia*, 2018.

[8] Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1534–1543, 2016.