FLiCR: A Fast and Lightweight LiDAR Point Cloud Compression Based on Lossy RI

Jin Heo
Georgia Institute of Technology
Atlanta, Georgia, USA
jheo33@gatech.edu

Christopher Phillips
Adeia
Hartwell, Georgia, USA
chris.phillips@adeia.com

Ada Gavrilovska
Georgia Institute of Technology
Atlanta, Georgia, USA
ada@cc.gatech.edu

***Abstract*—Light detection and ranging (LiDAR) sensors are becoming available on modern mobile devices and provide a 3D sensing capability. This new capability is beneficial for perceptions in various use cases, but it is challenging for resource-constrained mobile devices to use the perceptions in real-time because of their high computational complexity. In this context, edge computing can be used to enable LiDAR online perceptions, but offloading the perceptions on the edge server requires a low-latency, lightweight, and efficient compression due to the large volume of LiDAR point clouds data.**

This paper presents FLiCR, a fast and lightweight LiDAR point cloud compression method for enabling edge-assisted online perceptions. FLiCR is based on range images (RI) as an intermediate representation (IR), and dictionary coding for compressing RIs. FLiCR achieves its benefits by leveraging lossy RIs, and we show the efficiency of bytestream compression is largely improved with quantization and subsampling. In addition, we identify the limitation of current quality metrics for presenting the entropy of a point cloud, and introduce a new metric that reflects both point-wise and entropy-wise qualities for lossy IRs. The evaluation results show FLiCR is more suitable for edge-assisted real-time perceptions than the existing LiDAR compressions, and we demonstrate the effectiveness of our compression and metric with the evaluations on 3D object detection and LiDAR SLAM.

***Index Terms*—lidar, lidar point cloud, lidar point cloud compression, 3D point cloud compression, remote lidar perceptions, real-time perception service, range image compression, edge computing**

I. Introduction

Light detection and ranging (LiDAR) sensors have been used in robotics and autonomous vehicles for robust and accurate 3D sensing. The 3D point clouds from LiDAR sensors are used in perception tasks for understanding real-world contexts, such as 3D object detection and tracking, and LiDAR simultaneous localization and mapping (LiDAR SLAM). A LiDAR sensor has advantages over image sensors as it can provide environmental information in 3D with higher accuracy and is more robust to challenging weather and light conditions. Previously, despite the benefits of LiDAR sensors, only a few device platforms were equipped with them because the early models of LiDAR were too expensive and large in size [1]. However, as LiDAR technology advances, the sensors are becoming smaller and affordable, while maintaining their sensing performance even at lower power usage. Recently, Velodyne Lidar released a palm-size LiDAR sensor, Velabit [2], for around $100, Intel RealSense camera [3] has a tiny LiDAR sensor, as do the latest Apple iPhone and iPad [4] devices. As LiDAR becomes cost-effective and smaller in size, there would be more opportunities for mobile devices to leverage perceptions of this new sense in diverse use cases.

Although LiDAR technology is becoming available on mobile devices, there remain challenges in leveraging LiDAR perceptions for real-time use cases. As discussed in prior LiDAR perception research [5]–[9], the computational complexities of LiDAR perceptions are very high because these algorithms process unstructured 3D data. Even with these highly optimized algorithms, high-end processors and GPUs are required to make them run in real-time. This poses limitations to enabling real-time LiDAR perception on resource-constrained (including by battery lifetime) mobile devices which lack such high-end hardware.

Edge computing is a technology that can relieve such issues and enable computationally intensive perceptions for mobile users with commodity hardware. The edge (or cloudlet) is located at the edge of the network and close to the end users in the multi-tier cloud. End users can utilize edge resources effectively for storing and processing data on nearby edge servers accessible via low-latency and high-bandwidth networks such as 5G [10]. There has been research on offloading image-based real-time perceptions on the edge [11]–[13]. When remotely running the perceptions, a sequence of images is transmitted to the edge server in real-time. In such settings, efficient and low-latency image compression is essential because of the large size of the raw images. Fortunately, image compression algorithms have been extensively studied by both industry and academia [14]–[16], and the accelerators for such standard codecs are broadly available even on mobile devices, given the popularity of video streaming [17]–[19]. These existing video codecs with accelerators enable the online perceptions on the remote edge servers.

While offloading image-based perceptions takes advantage of the available codecs and accelerators for real-time video streaming, there are challenges to enabling edge-assisted online LiDAR perceptions due to the lack of such infrastructure for LiDAR point cloud compression. LiDAR sensors generate unstructured 3D point clouds, and their volume is too large to send them as raw data. As an example, the points per scan of the KITTI dataset [20] are about 120,000 of 2 MB, and streaming the raw sensor data at 60 frames per second

(FPS) needs a bandwidth of 120 Mbps. In addition to the high bandwidth usage, transmitting a large amount of data not only causes higher network loads on the backend middleboxes, resulting in additional transmission delays, but also reduces the lifetime of a mobile device by consuming its battery [21]–[23]. In this context, an effective point cloud compression is crucial, but it is required to be low-latency and lightweight. Since the responsiveness of online perceptions is determined by the end-to-end latency, high compression latency can introduce discrepancy between the real environment and the perception results [24]. The compression time should be sufficiently small not to compromise the benefit of the reduced perception processing time on the server. Moreover, the compression should be lightweight to run on mobile devices.

There are prior efforts to improve 3D point cloud compression, but their primary focus is on achieving higher compression ratio while preserving the original content qualities [25]–[31]. Even with real-time compression methods, the latency ranges of previous methods are too high to enable online perceptions, or they require high-end processors with GPUs for low latency [32]–[37]. In short, the effectiveness of existing point cloud compression methods is limited when considering mobile devices, which poses a challenge to enabling edge-assisted LiDAR perceptions.

For enabling edge-assisted LiDAR perceptions to real-time applications, we propose **FLiCR**, a lightweight and low-latency point cloud compression method based on the range-image (RI) representation and a lossless compression algorithm. While previous research on RI compressions utilizes only the quantization of the point bit precision with lossless RI mapping [26]–[29], [32], [33], we explore the optimization opportunities of lossy RIs with subsampling for reducing the data size and improving the compression efficiency. The idea is that compression algorithms such as dictionary coding, which use shorter references to repetitive features, would become more effective since they operate with a more limited data representation space. Subsampling of mapped points leads to point loss, and it is criticial to understand how this translates to reduction in end-to-end perception quality. We demonstrate the limitations of existing quality metrics to represent this total information loss because their designs are only concerned with point-to-point distances. Then, we propose a unified metric, **ePSNR**, that captures both point-wise and entropy-wise point cloud qualities, by extending the current PSNR with a probability function of entropy estimation.

We evaluate FLiCR and ePSNR with the current compression methods and metrics and different LiDAR perception use cases. In our results, FLiCR achieves up to 5.3× improvement in end-to-end compression latency on mobile devices and 12.6× in compression ratio compared to Google Draco, and ePSNR captures the quality impact of the lossyness introduced by FLiCR, enabling future system support to dynamically exercise the latency-performance tradeoff it exposes. In summary, we make the following contributions:

- We identify the requirements of LiDAR point cloud compression methods for edge-assisted online perceptions and conduct a thorough analysis on the limitations of the state-of-the-art technologies.
- We propose FLiCR, a lightweight, low-latency, and efficient compression method that combines use of lossy RI and lossless dictionary coder, and compare it to the existing methods.
- We point out the limitations of the current quality metrics for point clouds in terms of the entropy loss and propose ePSNR as a new single-number metric reflecting both point-wise and entropy-wise qualities.
- We demonstrate the benefits of our compression method and metric on two downstream perception tasks, 3D object detection and LiDAR SLAM.

II. Background

3D Point Clouds. A 3D point cloud is a set of points in the 3D space. Point clouds can be categorized into two categories by their characteristics: structured and unstructured. The unstructured (raw) point cloud is a sequence of the coordinate values of 3D points (usually *x, y, z* in a Cartesian coordinate system), optionally with other attributes such as reflection intensities. The structured point cloud is a point set organized with geometric or hierarchical structure contexts including meshes, octrees, etc. A LiDAR point cloud is an unstructured point cloud directly captured from LiDAR sensors.

Unstructured Point Cloud Compression. There are diverse existing compression methods, but a common thread across them is to convert raw point clouds into structured intermediate representations (IRs) and apply compression algorithms to the IRs, as shown in Figure 1. The compression process is tied to each IR, and the commonly used IRs are k-d tree, octree, mesh, and range image. Figure 2 shows different IR visualizations from a raw point cloud. Compression methods are categorized into geometry-based or image-based compression, based on the used IRs. Geometry-based compression uses the tree structures or mesh [25], [30], [34], [38]–[42], and the image-based compression maps the point clouds into 2D frames [26], [27], [32], [33], [36]. The geometry-based compressions code their IRs and compress the coded IRs, and the image-based approaches utilize the existing codecs or present their own techniques for compressing the mapped images. More details of the existing methods appear in Section VIII.

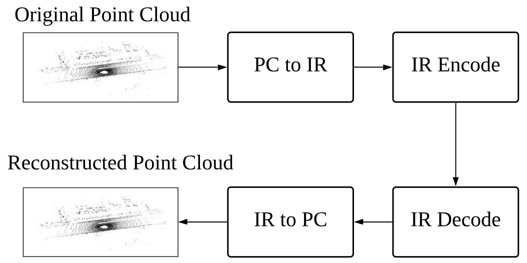
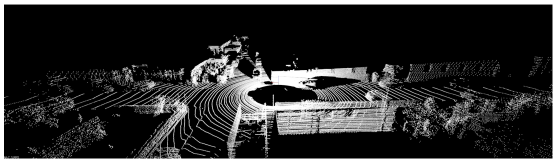
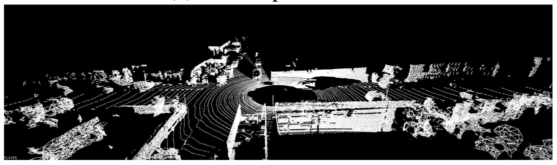


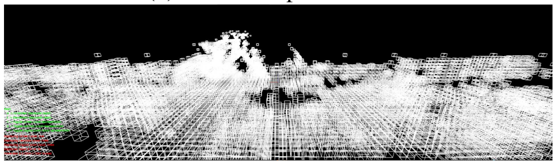
Fig. 1: The general processing steps of the existing point cloud compression methods.



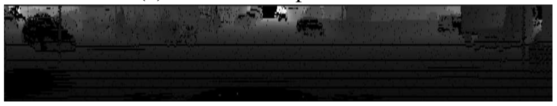
(a) A raw point cloud.



(b) A mesh representation.

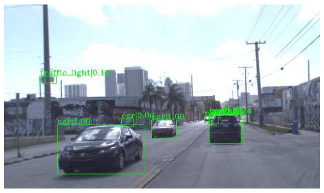


(c) An octree representation.

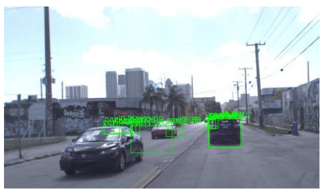


(d) A range image.

Fig. 2: The visualizations of a LiDAR point cloud from KITTI [20] dataset with different IRs.



(a) The object detection result without the discrepancy latency.



(b) The object detection result with the discrepancy latency of 300 ms.

Fig. 3: The simulated results of the online perceptions with Mask R-CNN [43] and Argoverse [45].

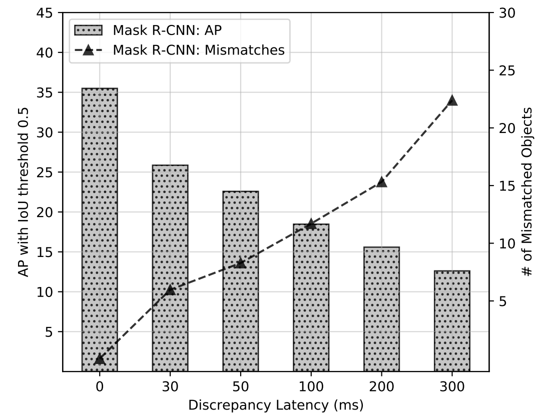


Fig. 4: The measured AP and number of mismatches of the online object detection with different discrepancy latencies.

III. Challenges with LiDAR Point Cloud Compression

Reducing discrepancy latency. For online perception, the end-to-end latency of the processing pipeline has a major impact on performance as it affects the application responsiveness to changes in the real-world environment [24]. For instance, if the perception result of the data captured at t_0 is available at t_1, there would be a discrepancy between the result and real world with the changes during the time from t_0 to t_1. Figure 3 is the screenshot of the simulated online perception results with and without 300 ms of the discrepancy latency in the object detection task. Without the discrepancy latency, all real-world objects are aligned with the detection results as Figure 3a. However, with the latency, the perception results are not correctly corresponding to the real-world objects because of the discrepancies as Figure 3b.

In Figure 4, we show the impact of different discrepancy latencies between the real world and the perception results on the performance of object detection. We use the metrics for streaming perceptions [24] – the average precision (AP) with intersection over union (IoU) threshold 0.5, and the number of mismatched objects between the results with and without discrepancy latencies. For object detection, we use Mask R-CNN [43] pre-trained with the dataset of Microsoft COCO [44]. Then, the detection model runs with an autonomous driving dataset, Argoverse [45], and we measure the metrics with different discrepancy latencies. Without the discrepancy latency, the perception model detects $\sim$36% of objects to the ground truth. As the latency increases, the AP result starts to decrease with the increased number of mismatched objects. These results show the latency governs the perception performance and there is the latency-performance tradeoff of online perceptions.

In addition to the perception algorithm, there are additional components in the processing pipeline which introduce latency when offloading the LiDAR perceptions on the edge: data compression and network transportation. The overheads from these steps contribute to the discrepancy latency, and the benefit of the reduced processing time of a perception can be compromised by them. As shown in Figure 4, even with 30 ms of discrepancy latency, AP decreases by $\sim$28% of the result without the latency. With 100 ms delay, it becomes half of the result without delay. Furthermore, the number of mismatched objects soars with the higher latencies. So, along with the application of edge computing to reduce the network latency, a lightweight and low-latency LiDAR point cloud compression method is essential to enable edge-assisted online LiDAR perceptions for resource-constrained mobile users.

Limitations of existing compression methods. Given the popularity of 3D point cloud data, there are existing technologies for LiDAR point cloud compression: Google Draco [42], MPEG Geometry based point cloud compression (G-PCC) [25], Point Cloud Library (PCL) [38] octree compression, and the real-time spatio-temporal (RT-ST) com-

TABLE I: The benchmark results of the existing compression methods for 3D point clouds. The results in parentheses are on the Jetson AGX.

Metrics \ Methods	RLE	Dict Coding [46]	Google Draco [42]	MPEG G-PCC [25]	PCL [38]	RT-ST [32]
Compression Ratio	0.54	1.67	**17.05**	8.76	5.72	15.96
PSNR (dB)	**Lossless**	**Lossless**	67.29	78.43	89.77	63.18
CD (cm)	**Lossless**	**Lossless**	0.267	0.184	0.001	3.07
Enc Time (ms)	40.1 (42.4)	40.5 (75.4)	**21.1** (**48.4**)	598 (741)	72.1 (198)	97.7 (240)
Dec Time (ms)	17.9 (15.8)	13.8 (33.7)	**9.44** (**18.6**)	204 (265)	55.4 (153)	15.2 (34.8)
Enc Energy Usage (J)	1.18 (**0.11**)	1.19 (0.23)	**0.83** (0.14)	15.35 (2.41)	2.05 (0.46)	2.63 (0.59)
Dec Energy Usage (J)	0.51 (**0.04**)	0.39 (0.08)	**0.36** (0.05)	5.56 (0.66)	1.54 (0.36)	0.57 (0.12)

pression by Feng *et al.* [32]. Google Draco is based on k-d tree, PCL and G-PCC are on octree, and RT-ST compresses range images (RIs) of 3D point clouds. Moreover, with these methods, it is possible to apply bytestream compressions of run-length encoding (RLE) and dictionary coding (LZ77) [46] to the point clouds directly; they treat point clouds as raw byte arrays and deflate them losslessly. We compare these methods based on performance, quality, and efficiency metrics on a desktop and on NVIDIA Jetson AGX of our experimental testbed, as described in Section VII-A. As the quality metrics of the point clouds, we use peak signal-to-noise ratio (PSNR) and Chamfer Distance (CD) as defined in Section VI. For each method, we encode and decode 100 point clouds from the KITTI dataset [20]. The averaged results are in Table I, showing the results of the desktop, and in parentheses the results from the Jetson.

While every method has its advantages for different metrics, we focus on the compression ratio, energy usage and latency, because these metrics show how well a compression method meets the requirements of point cloud compression for edge-assisted online perceptions. For our target use case, there are three requirements of a compression method.

(1) It should be very low-latency because of the latency-performance tradeoff of online perceptions;
(2) The compression performance effectiveness in reducing the data size is important since the larger size of compressed data causes higher network cost and energy consumption on the client to transmit;
(3) It should be lightweight to run on mobile devices of limited resources while satisfying the other requirements.

For the compression ratio and latency, Google Draco outperforms the other methods and is highly efficient in terms of the compression ratio and energy usage. Although RLE shows comparable latency and energy usage on Jetson, it shows increased total size when applying RLE directly to the floating-point values of 3D points. Except for the lossless methods, PCL's octree compression shows the highest quality metrics. However, it is at the cost of the lower efficiency and high energy usage versus Google Draco.

Based on the results in Table I, Google Draco seems the best option to meet the requirements, but it causes about 60 ms ($\sim$50 ms for encoding and $\sim$10 ms for decoding) of the compression cost when the user device is Jetson and our desktop is a server. Since there are additional delays from network transmission and algorithm processing, the discrepancy latency of the whole pipeline will be too high given a compression cost of 60 ms. This would hurt the perception performance and be hard to use, as illustrated in Figure 4.

In summary, *there is a need for a low-latency, lightweight, and efficient LiDAR compression method for edge-assisted online perceptions*, which motivates us to pursue this work.

IV. Intermediate Representations for FLiCR

For meeting the aforesaid requirements, it is important to select a proper IR because the compression is dependent on each IR. In this section, we microbenchmark the IR conversions and point out the benefit of range images (RIs) over the others in the context of enabling remote online perceptions.

TABLE II: The latencies (ms) of each IR construction.

	RI	Parallel RI	Octree	K-d tree	Mesh
Desktop	11.78	**6.72**	30.67	13.21	1872
Jetson	16.34	**9.26**	32.11	32.44	2755

Table II shows the conversion latencies of the IRs with the LiDAR point clouds from the KITTI dataset [20]. We use PCL [38] implementations for octree and k-d tree, and mesh conversion is based on the algorithm of Marton *et al.* [47]. The RI conversion is our implementation, and the parallelized version is with OpenMP [48]. The RIs are generated by converting the raw points in the 3D Cartesian coordinates to the spherical coordinates. Equation 1 shows the conversion and r, θ, and ϕ are the radial distance, polar angle, and azimuthal angle each. When θ and ϕ are calculated, they are mapped to the frame pixel by the sensor's angular precisions. For example, Velodyne HDL-64E used in the KITTI dataset has 0.08° and 0.35° for horizontal and azimuthal precisions with 360° of the horizontal field of view (FoV) and 64 vertical lasers [49]. Thus, each scan's point cloud would be mapped to a 2D frame of 4500×64. By adjusting the parameters of precisions and FoV, RI can work on diverse LiDAR sensors.

$$\begin{aligned} r &= \sqrt{x^2 + y^2 + z^2} \\ \theta &= arccos\left(\frac{z}{r}\right) \\ \phi &= arctan\left(\frac{y}{x}\right) \end{aligned} \tag{1}$$

In our results, the parallelized RI conversion shows the lowest latency on both desktop and Jetson as RI has advantages over other IRs in terms of simplicity and parallelism. For the octree and k-d tree, there have been many efforts for their parallelized constructions [50]–[54]. However, as pointed out in the previous works, their hierarchical structures inherently make the construction processes sequential, and it is challenging to fully parallelize their constructions. In contrast, the RI conversion can be easily parallelized since each point conversion of RI is completely independent from the others. For the mesh, its generation from point clouds requires triangulation algorithms and calculating the surface normal for each mesh. These processes require iterating each point and finding nearest neighbors to generate a mesh, and the mesh conversion has high computational complexity and is not suitable for real-time due to its large magnitude of execution time [47], [55], [56].

Furthermore, these IRs have different theoretical complexities for the conversion. The time complexities of the IR constructions are $O(n)$ for the RI and $O(n \log n)$ for the trees and mesh; the trees require binary search for each point insertion, and the mesh construction needs nearest neighbor searches for the normal estimation and triangulation. In addition, there is a side benefit that various image-processing techniques can be used for RIs. Based on these observations, we adapt RI as the target IR.

V. FLiCR: Range Image Compression

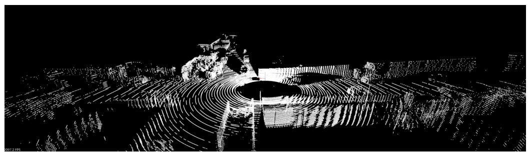
Following selecting RI as the appropriate IR, the compression also needs to be efficient, low-latency, and lightweight. In this section, we describe how to achieve the objectives of the compression method. First, we identify the distortion issue of the current image codecs to LiDAR RIs. Second, we explore the opportunity of lossy RIs for the downstream compression steps via RI quantization and subsampling. We argue the lossless bytestream compressions can be hugely enhanced in terms of the compression efficiency and low latency through the lossy representation. However, it compromises the point cloud quality, and we present the possible issues of FLiCR with lossy RIs.

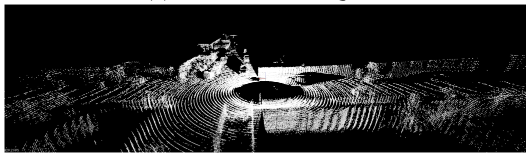
TABLE III: The compression ratios and qualities of H.264 with different QPs for 100 RIs of 4500×64 and 8 bpp.

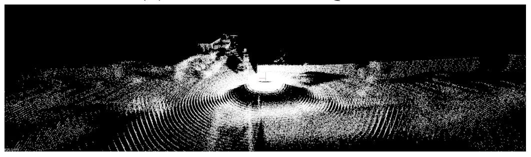
	QP 0	QP 10	QP 20	QP 30
Compression Ratio	12.85	13.33	16.41	35.01
PSNR (dB)	63.18	48.21	37.61	38.12
CD (cm)	2.23	16.25	126.06	107.16

A. Issues with Current Image Compressions

By representing LiDAR point clouds as images, it becomes possible to leverage the existing image-processing infrastructures and techniques. With the popularity of video streaming, modern processor platforms and GPUs are equipped with dedicated hardware modules for the standard codecs such as H.264 and HEVC [17]–[19]. These codecs efficiently encode and decode continuous images with spatial and temporal optimizations [14], [16], and the pervasive accelerators enable the codecs in a low-latency and efficient way even with commodity mobile devices.


(a) Point cloud of QP 0.


(b) Point cloud of QP 10.


(c) Point cloud of QP 20.

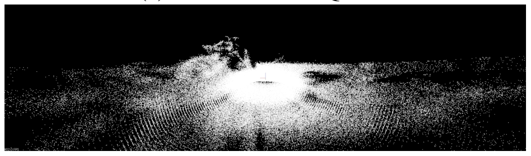

(d) Point cloud of QP 30.

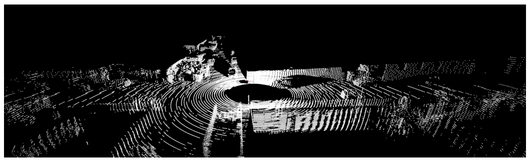
Fig. 5: The visualizations of the reconstructed point clouds from the RI of 4500×64 with H.264 and four different quantization parameters.

In this context, it seems appropriate to rely on the existing codecs with hardware accelerators at first glance. However, we argue the existing codecs specialized for human vision are hardly applicable to RI compression. The lossy image compression algorithms for human vision fully utilize the characteristics of the human eyes to remove the data with minimal impacts to visual quality as much as possible; one example is to convert an image into the frequency domain via discrete cosine transform (DCT) or fast Fourier transform (FFT) and the high frequency has more loss than the low-frequency data [14], [16], [57]. While the techniques that leverage the nature of human vision work well for normal images, the point cloud details are effectively lost as a result of the frequency-domain loss in LiDAR RIs.

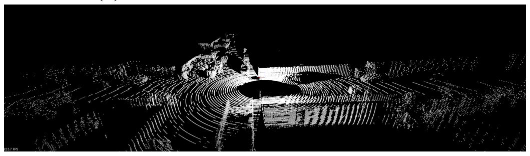
Figure 5 shows the reconstructed point clouds from the RIs encoded and decoded via H.264 with different quantization parameters (QP). QP regulates how much spatial detail is retained and is set from 0 for lossless to 51 for the most lossy compression. As QP increases, the spatial detail is aggregated so that the encoded bit rate drops at the expense of data loss, resulting in lower quality [14]. The reconstructed point clouds become vague and noisy with high QPs. Table III shows the averaged compression ratio and quality metrics of the reconstructed point cloud with different QPs. With the visual results of the reconstructed point clouds, the PSNR and CD results become worse drastically while the compression ratio increases moderately. Considering that the quantization parameter such as QP or CRF of video streaming is usually set around 20 and 30 as a rule of thumb (FFmpeg's H.264 default CRF is 23 [58]), these results show the current human-vision codecs are unsuitable for the RI compression. For preserving the quality of point clouds, the codec quantization parameter should be set for lossless (QP 0), but it is at the cost of the lower compression efficiency, as Google Draco achieves ~33% higher compression ratio in Table I.

TABLE IV: The existing quality metrics with sampling error (SE) for the subsampled RIs of 8 bpp.

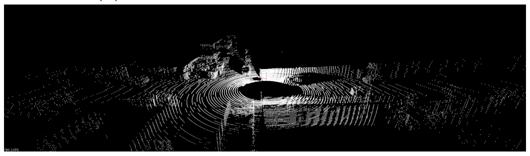
	2048×64	1024×64	512×64	256×64
PSNR (dB)	62.4	61.41	58.61	53.71
CD (cm)	5.37	9.23	15.22	36.17
SE	21.03%	58.77%	78.95%	89.22%



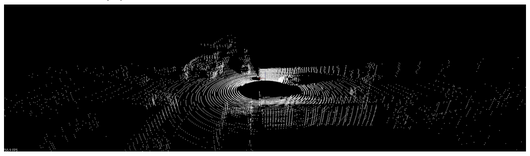
(a) Point cloud from 2048×64 RI.



(b) Point cloud from 1024×64 RI.



(c) Point cloud from 512×64 RI.



(d) Point cloud from 256×64 RI.

Fig. 6: The visualizations of the point clouds reconstructed from the subsampled RIs of Figure 2a corresponding to 4500×64 RI.

B. RI Quantization and Subsampling

RI has been used for losslessly mapping LiDAR point clouds to 2D frames, and previous work only applies quantization of bit-per-point (bpp) [26]–[29], [32], [33]. In these prior works, the main objective is to maximize the compression efficiency while maintaining the point cloud quality as high as possible. However, we argue that there are more optimization

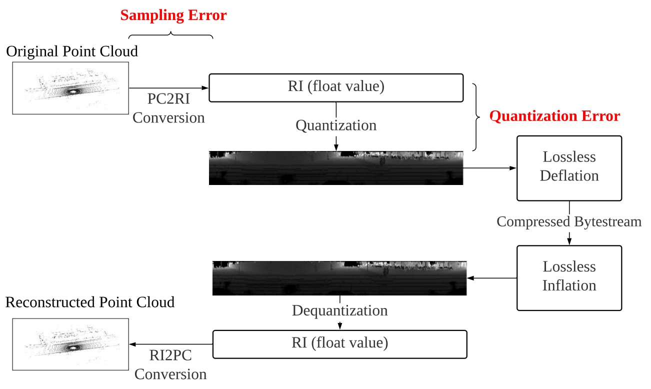


Fig. 7: FLiCR compression steps with lossy RIs of subsampling and quantization and lossless compression algorithms.

opportunities with lossy RI to decrease not only the data size but the downstream compression tasks' complexities. Specifically, the RI resolution is determined by the sensor's precisions as mentioned in Section IV, and the subsampling of point clouds can be done by adjusting the precision parameters; the 3D points are coarsely mapped to a 2D frame. Figure 6 shows the visualizations of reconstructed point clouds from the subsampled RIs. From the raw point cloud of Figure 2a, we reduced the precision parameters to map it to the RIs of four different lower resolutions. Even with the lowest subsampled RI of 16 KB with 8 bpp, the shapes of scanned objects are recognizable.

While the subsampled and quantized RI has advantages for data reduction and compression with lower latencies, it would affect the performance of the perception tasks. So, a quality metric for the point clouds from lossy RIs needs to reflect both the quantization and subsampling errors. The currently used metrics, PSNR and CD, reflect the quantization error well, but the sampling error of lossy RIs is not represented effectively as these metrics are defined with the point-to-point distances between point clouds (see Section VI for more details).

Table IV shows PSNR, CD, and sampling error (SE) of the point clouds from four RI resolutions. In the results, the changes of PSNR and CD exhibit different trends from SE because SE is about the number of lost points from the original point cloud (the entropy-wise quality) while PSNR and CD are with the distances of the closest point pairs between two point clouds (the point-wise quality).

The current metrics' issue is they only count the point-to-point distances, and each point distance is calculated by finding the nearest point in the comparing point cloud. So, when the point clouds have different numbers of points, they are limited to represent this difference in the total number of points in the point clouds. To address the limitations of the existing metrics, we propose a unified metric for both the point-wise quality and the information amount to measure quantitatively the impacts on the downstream perceptions from lossy RIs in Section VI.

C. Lossless Compression with Lossy RIs

As shown in Section V-A, the application of lossy video codecs to RIs results in lower compression efficiency or can distort the point clouds in the 3D space. The previous RI compression methods apply the image compression algorithm at lower efficiencies or propose effective lossless RI compression algorithms via spatial and temporal optimizations [26]–[29], [32], [33]. However, they partially leverage the opportunities of lossy RIs only with bit quantization, and their complex algorithms have downsides in terms of latency and overheads, while showing high compression ratios. For satisfying the low-latency, lightweight, and efficiency requirements, we use the existing bytestream compression algorithm, dictionary coding, and enhance its efficiency by fully leveraging the RI quantization and subsampling.

Dictionary coding is a lossless compression algorithm for bytestreams and deflates the bytestream by replacing the repeating patterns with shorter references. Dictionary coding algorithms have been extensively studied with corpus text data, and they are with simple bit/byte operations and lower computation complexities in terms of the space-time trade-off [59]. So, they provide the benefits of being lightweight and low-latency, with simple operations and do not distort point clouds unexpectedly. Even with such advantages, the direct application of bytestream compressions to the raw point cloud and unquantized RIs of floating values is inefficient in terms of the compression ratio (RLE and Dict Coding in Table I and Figure 8a).

To improve the efficiency, we fully utilize both quantization and subsampling. The underlying assumption of our approach is dictionary coding uses the repeating features in a bytestream and there is a higher probability of recurring patterns when limiting the representation space of quantized RIs. The compression pipeline is shown in Figure 7. FLiCR with dictionary coding has similarity to previous RI-based works in terms of leveraging local spatial features, but it is more advantageous in meeting the requirements with its simplicity. Explicitly, compared to the recent RI-based compression (RT-ST [32] in Table I), FLiCR shows lower encoding and decoding latencies and energy usage, with the higher compression ratio, as shown in Table V.

Among the dictionary coding algorithms, we use LZ77 [46] and compare it with RLE. We measure the efficiency improvement and quality reduction by the quantization and subsampling with LZ77 and RLE, and Figure 8 shows the results of different resolutions of RIs quantized by 8 bpp. While the end-to-end latencies of the whole compression pipeline can decrease only with subsampling (see Figure 8b), both quantization and subsampling are required to improve the compression ratios effectively, as shown in Figure 8a. Then, dictionary coding shows larger growth than RLE, and these results support our assumption about the performance improvement of dictionary coding with lossy RIs.

Although FLiCR achieves compression efficiency and reduced latency, it is at the cost of degradation of the point cloud quality by the quantization and subsampling errors, as shown in Figures 8c and 8d. Since the reduced point cloud quality can have an impact on the downstream perceptions, we evaluate FLiCR with the state-of-the-art LiDAR perceptions and analyze the errors' impacts in Section VII.

Figure 9 shows the latency breakdowns of FLiCR on our testbed. In the case where Jetson is a mobile client and the desktop is a server, the end-to-end latency is $\sim$39 ms ($\sim$60% of Google Draco [42]) even with the highest RI resolution; it takes 27 ms for client encoding and 12 ms for server decoding. With the 256$\times$64 resolution, the end-to-end latency is $\sim$10 ms which is $\sim$16% of Draco. Since the large portion of the end-to-end latency is the conversion time between the point cloud and RI, the end-to-end latencies can be largely reduced if a device has a dedicated hardware logic for the RI conversion. As the RI resolution gets lower, the quantization and compression latencies decrease. These results show FLiCR fully leverages the synergistic effect by quantization and subsampling for the bytestream compressions.

VI. ePSNR: Quality Metric for LiDAR Point Clouds

The errors in the RI conversion process can affect the performance of LiDAR perceptions. PSNR and CD (or RMSE) have been broadly used as the quality metrics of 3D point clouds [26], [27], [30], [32]–[34], [40], but by definition they are point-wise quality metrics and do not reflect the point loss effectively. In the context of our approach with lossy RIs, we argue a metric for both point-wise quality and overall information amount is essential.

$$Dist(p, C) = \min_{p_c} \left((p_c - p)^2 \right) \tag{2}$$

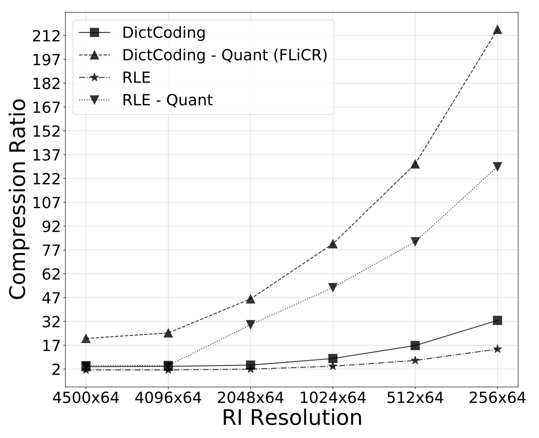
$$MSE(C_1, C_2) = \frac{1}{\|C_2\|} \sum_{i=0}^{\|C_2\|-1} \{Dist(p_{c_2}, C_1)\} \tag{3}$$

Both PSNR and CD use the mean squared error (MSE) of the point-wise distances between two point clouds. When C_1 is the original point cloud and C_2 is the reconstructed point cloud, the distance between a point in C_2 and the corresponding point in C_1 is calculated by Equation 2. So, the corresponding point in C_1 is of the shortest distance to the point in C_2. Then, MSE between two point clouds is defined by Equation 3.

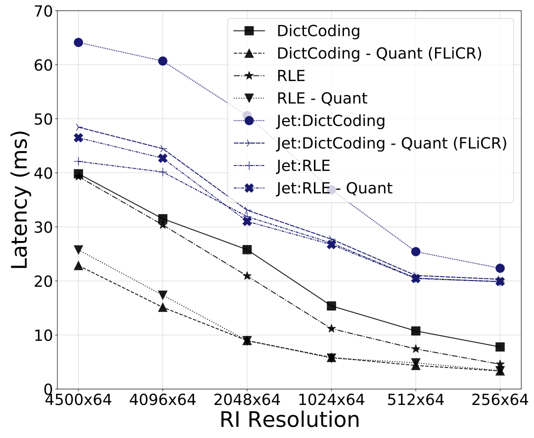
$$CD(C_{orig}, C_{comp}) = MSE(C_{orig}, C_{comp}) + MSE(C_{comp}, C_{orig}) \tag{4}$$

$$PSNR(C_{orig}, C_{comp}) = 10\ log \left(\frac{Max^2}{MSE(C_{orig}, C_{comp})} \right) \tag{5}$$

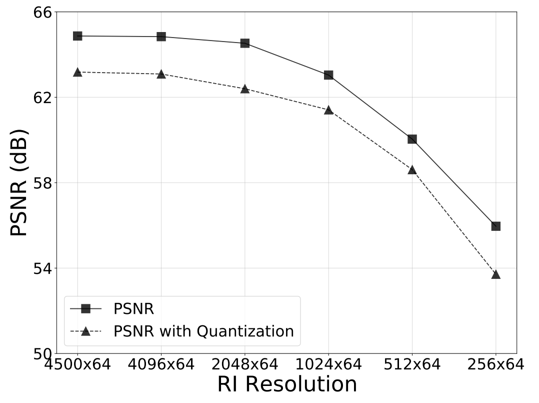
Then, PSNR and CD are defined as Equation 4 and 5. CD is the sum of reciprocal MSEs between two point clouds, and PSNR is the ratio of the peak LiDAR sensor range to MSE. As indicated by their definitions, these metrics are based on



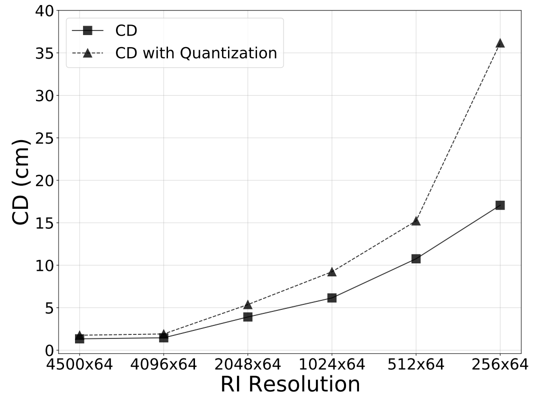
(a) The compression ratios with and without quantization in different RI resolutions.



(b) The end-to-end latencies with and without quantization in different RI resolutions.



(c) The PSNR results with and without quantization in different RI resolutions.



(d) The CD results with and without quantization in different RI resolutions.

Fig. 8: The quantization and subsampling impacts on the lossless bytestream compressions.

MSE and are determined by the point-to-point distance. They natively represent the quality loss by the quantization error, but the subsampling error is not effectively represented even if the impact of SE is shown mildly as some nearest points can be lost in the reconstructed point cloud. Specifically, in the current metrics, it is possible to get a high-quality result even with a few points of small distances to a point cloud of large number of points. It is caused by the unstructured nature of the LiDAR point cloud; there is no point-to-point correspondence between LiDAR point clouds having different numbers of points. In the case of the normal images, the total number of pixels is fixed without SE and the point-wise metrics work well.

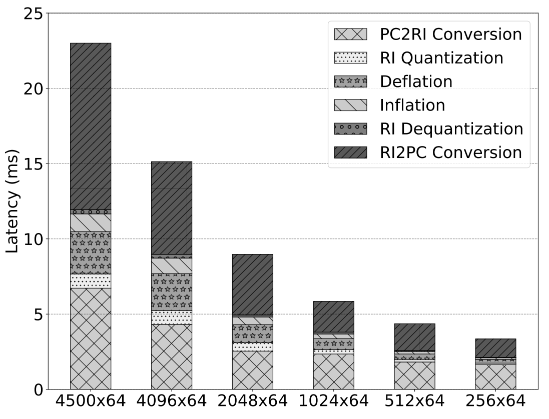
$$SE = \frac{\|C_{orig} - C_{comp}\|}{\|C_{orig}\|},\ 0 \leq SE \leq 1 \tag{6}$$

Based on our observation, we argue it is inappropriate to use the metrics representing only the point-wise quality for LiDAR point clouds. To address the limitation of the current metrics, we propose a new single-number metric, **entropy-reflecting PSNR (ePSNR)**, by extending PSNR. ePSNR is designed to indicate both the point-wise and entropy-wise quality of a point cloud.

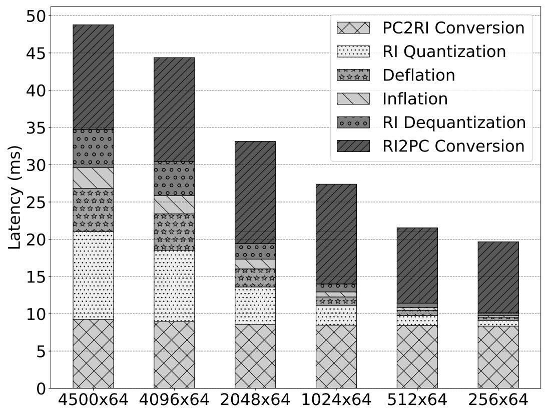
SE is related to the total information (entropy) loss in a point cloud because it is the percent of the lost points, as in Equation 6. One naive way of making PSNR reflect the entropy is to multiply $1-SE$ to PSNR with the assumption that the entropy is $1-SE$ and SE is exactly the same with the actual entropy loss, $\mathscr{L}_{SE}$. However, instead of the naive way, we extend PSNR by estimating $\mathscr{L}_{SE}$. Our underlying assumption is $\mathscr{L}_{SE}$ is not exactly the same with SE and follows the exponential distribution as Equation 7. The intuition for this assumption is that SE can have minimal impacts on the downstream perceptions as far as the total amount of necessary information is preserved for the perception algorithms. It means there would be a knee of the curve in the graph of the entropy function.

$$Assumption: \quad \mathscr{L}_{SE} \sim \exp(\beta) \tag{7}$$

When $\mathscr{L}_{SE}$ follows the exponential distribution, the entropy function $\mathcal{F}(SE)$ can be defined with the cummulative distribution function (CDF) of the exponential distribution as Equation 8. This entropy function is a probability function estimating the actual entropy of the remaining points in a point cloud with the given SE.



(a) The latency breakdown of our method on desktop.



(b) The latency breakdown of our method on Jetson AGX.

Fig. 9: The end-to-end latency breakdowns of FLiCR.

$$\mathcal{F}(SE) = \mathrm{P}(\mathrm{E} > x) = e^{-\frac{x}{\beta}} \quad where\ x = 1 - SE \tag{8}$$

With our entropy function, ePSNR is defined as Equation 9. Since it is based on PSNR, the point-wise quality with the quantization error is represented while reflecting the entropy with the given SE. When SE is small, ePSNR would be almost same with the original PSNR, but would start to decrease exponentially when SE gets larger, by its definition. ePSNR has two parameters: α as a derivative adjusting factor to prevent too steep or shallow distribution and β of the exponential distribution.

$$\begin{gathered} ePSNR(C_{orig}, C_{comp}) = PSNR \times \{1 - (SE \times (\mathcal{F}(SE) + \alpha))\}, \\ 0 \leq \mathcal{F}(SE) + \alpha \leq 1 \end{gathered} \tag{9}$$

VII. Evaluation

The goal of this section is to demonstrate that our approach appropriately meets the requirements of the LiDAR point cloud compression for enabling edge-assisted online perceptions. We compare FLiCR with several existing compression method. Since FLiCR affects the quality of the point clouds, its impact is evaluated with the state-of-the-art LiDAR perception algorithms for 3D objection detection and LiDAR odometry and mapping (LOAM). We also evaluate ePSNR and demonstrate its effectiveness compared to PSNR and the naive way of combining PSNR and SE. All experiments are done by using the LiDAR point clouds of the KITTI dataset [20], which are captured from Velodyne HDL-64E [49].

A. Experimental Testbed

The testbed consists of two machines, an NVIDIA Jetson AGX Xavier and a high-end desktop. The Jetson has ARMv8 CPU and 32 GB memory, and we set its power mode as 15W. The desktop has Intel Core i7-10700, 32 GB memory, and NVIDIA RTX 2070 GPU. Both run Ubuntu 18.04, and the energy usage is measured with `perf` on desktop and `tegrastats` on Jetson.

B. FLiCR Benchmark

As shown in Table I, Google Draco [42] is the most suitable compression method for online perceptions in terms of compression ratio, latency, and energy usage. So, we compare FLiCR with different RI resolutions to Draco. We benchmark FLiCR and measure the compression ratio, SE, PSNR, ePSNR ($\alpha = -0.15$, $\beta = 0.5$), latencies, and energy usage. The compression level parameter of Draco is set as 10 which is maximum. For FLiCR, each RI is quantized by 8 bpp. The benchmark results are shown in Table V.

FLiCR achieves higher compression ratios across all resolutions than Draco, and it is $\sim$25% higher even with the highest resolution. As highlighted in Table V, FLiCR with little subsampling starts to outperform Draco in the compression ratio, latency, and energy usage. One characteristic of Draco is its encoding process takes longer and uses more energy than decoding while with FLiCR it is similar in most cases. Considering the use case of edge-assisted perceptions, it is the client who encodes and sends data, and the server receives and decodes the encoded data to feed it to perception modules. When the client is Jetson and the server is the desktop in our testbed, Draco introduces $\sim$60 ms end-to-end latency, and with FLiCR it ranges from $\sim$11 ms for 256$\times$64 to $\sim$40 ms for 4500$\times$64. Therefore, FLiCR is more advantageous for the latency-performance tradeoff of online perceptions for commodity mobile devices, given its higher compression efficiency in terms of compression ratios and energy usage.

To satisfy the aforementioned requirements, we compromise the quality of point clouds with lossy RIs. Since Draco quantizes each point as 11 bpp, it shows a higher PSNR, and ePSNR is almost same with PSNR as it has small SE. One issue about SE is that the highest resolution RIs have $\sim$8% SE even though its resolution is with the maximum sensor precision of HDL64 specified in the spec sheet [49]. We presume this SE is caused by the sensor's measurement error and noise as the LiDAR point clouds are captured by running cars. In Table V, PSNR barely changes with 1024$\times$64 RIs which have 58.8% SE while ePSNR reflects it. Since the reduced quality affects the performance of downstream

TABLE V: The comparison between Google Draco and FLiCR of different RI resolutions.

	Google Draco	FLiCR 4500×64	FLiCR 4096×64	FLiCR 2048×64	FLiCR 1024×64	FLiCR 512×64	FLiCR 256×64
Compression Ratio	17.05	21.26	**24.75**	46.18	80.88	131.13	215.85
SE	**6.7%**	8.4%	9.1%	21%	58.8%	78.9%	89.2%
PSNR (dB)	**67.29**	63.18	63.09	62.4	61.41	58.61	53.71
ePSNR (dB)	**67.27**	63.13	63.01	61.64	51.38	35.4	22.29
Enc Time (ms)	21.1 (48.4)	10.48 (26.83)	**7.69** (**23.41**)	4.3 (16.03)	3.37 (12.26)	2.35 (10.46)	1.99 (9.54)
Dec Time (ms)	9.44 (18.6)	12.52 (21.94)	**7.44** (**20.97**)	4.67 (17.09)	2.47 (15.13)	2.01 (11.06)	1.36 (10.11)
Enc Energy Usage (J)	0.83 (0.14)	0.36 (0.09)	**0.27** (**0.07**)	0.16 (0.04)	0.13 (0.03)	0.09 (0.03)	0.07 (0.02)
Dec Energy Usage (J)	0.36 (0.05)	0.48 (0.05)	**0.3** (**0.05**)	0.19 (0.04)	0.13 (0.04)	0.09 (0.03)	0.08 (0.02)

perceptions, we also evaluate our compression and metric with the state-of-the-art LiDAR perceptions.

C. End-to-end Evaluation

We evaluate our method and metric with two perception tasks: 3D object detection and LOAM. For 3D object detection, we use machine learning (ML) models pre-trained with the original point clouds from the KITTI dataset from the Model Zoo of OpenPCDet [60]. We use the following models: Part-A^2 Net [61], PointPillars [62], PointRCNN [63], PV-RCNN [64], SECOND [65], Voxel R-CNN [66]. These models are trained with 7481 samples, and the testset is 7518 LiDAR scans. For LOAM [67], we use the A-LOAM implementation [68]. For checking the impacts of RI quantization and subsampling, we generate the LiDAR point cloud dataset reconstructed from different resolution RIs. Then, we feed our dataset to those perception models. Since the object detection models are trained with the original LiDAR data and A-LOAM is implemented and tested by using the original dataset, we can quantitatively measure the impacts of lossy RIs in FLiCR on the perception performance.

3D Object Detection. 3D object detection is the task of detecting objects from 3D point clouds. Each algorithm of the models we use has a different network architecture, but there is a commonality between them: a backbone network extracts features from the point clouds and the extracted features are used by the regional proposal networks (RPN). As the backbone networks of these models, PointNet++ [69] is used to extract the point-level features. For the voxel-level features, the voxel feature encoder (VFE) layer and 3D sparse convolutional networks [70] are used. When each model produces the region proposals of the detected objects, they are compared with the region of the ground truth objects. The result recall is determined by the detected objects corresponding to the ground truth object with the IoU threshold.

Table VI shows the recall for the detected objects of the models. With the highlighted example of PointPillars, the performance reductions between the original and 4500×64 RIs show the impact of the quantization error. In the case of IoU threshold 0.7, it shows $\sim$23% performance reduction while it is $\sim$2% with threshold 0.3. This shows the results of higher IoU thresholds are more sensitive to the quantization error, and 3D object detection with a higher IoU threshold requires input point clouds of almost the same quality as the original training data.

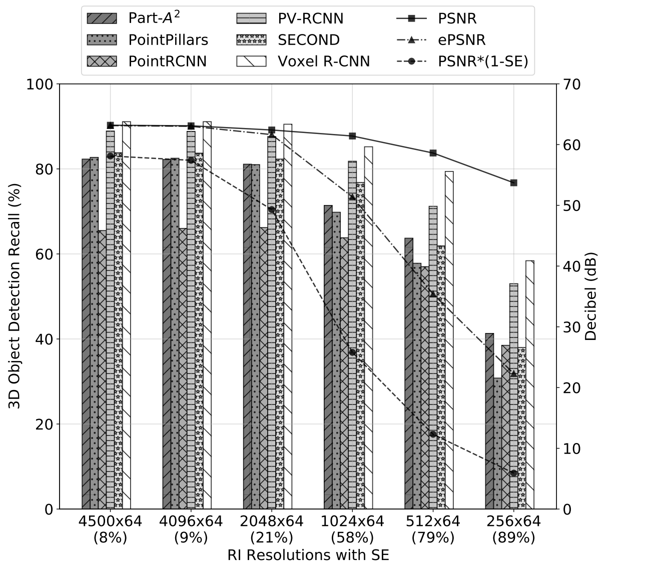


Fig. 10: The results of PSNR, ePSNR, and the naively-entropy-reflecting PSNR with the 3D object detection results of the IoU threshold 0.5 for the models.

The results across the different resolutions show the impacts of SE. One noticeable thing is the performance results decrease little with 2048×64 RIs compared to 4500×64 RIs, and this trend is for all IoU thresholds. These results support our assumption for ePSNR in Section VI; there is a knee of the curve in the entropy loss by SE. Moreover, Table V shows the ePSNR results drop drastically from 1024×64 RIs as does the performance of the 3D objection detection models.

Figure 10 shows the object detection recall values, and PSNR, ePSNR, and the naive way of making PSNR capture entropy, PSNR$\times(1-SE)$, as described in Section VI. The IoU threshold is 0.5 for all detection models, and the parameters of ePSNR are α (-0.15) and β (0.5). For the changes of SE and recalls, PSNR mildly changes across the RI resolutions, and PSNR$\times(1-SE)$ shows more drastic decreases compared to the perception results. On the other hand, ePSNR shows a similar trend with the performance reduction of the perception models. These results demonstrate the effectiveness of ePSNR with the probability function estimating the actual entropy by using SE, as a single-number metric for the point-wise and entropy-wise qualities of a point cloud.

LiDAR Odometry and Mapping. LOAM (or LiDAR SLAM)

TABLE VI: The 3D object detection performances with different IoU threshold and reconstructed point clouds from the RIs. The number in each cell is the recall for the detected objects in the scene.

		Original	4500×64 RI	4096×64 RI	2048×64 RI	1024×64 RI	512×64 RI	256×64 RI
IoU Threshold 0.3	Part-A^2 Net	95.1	88.4	88.3	87.9	76.2	75.5	56.1
	PointPillars	**94**	**92.6**	**92.4**	**91.5**	**82**	**76.1**	**54.8**
	PointRCNN	89.8	70.7	71.2	71.7	68.8	62.9	47.5
	PV-RCNN	96.8	94.6	94.4	94	89.9	82.8	70.2
	SECOND	94.9	92.7	92.6	92.1	88.4	79.4	60.2
	Voxel R-CNN	95.4	93.6	93.6	93.5	89	87.5	76.1
IoU Threshold 0.5	Part-A^2 Net	91.2	82.3	82.2	81.1	71.4	63.7	41.3
	PointPillars	**88.7**	**82.7**	**82.5**	**81**	**69.8**	**57.8**	**30.8**
	PointRCNN	87.1	65.5	66	66.2	63.8	57	38.5
	PV-RCNN	93.4	88.9	88.8	87.6	81.8	71.2	53
	SECOND	89.1	83.8	83.7	82.3	76.8	61.9	38
	Voxel R-CNN	94.9	91.1	91.1	90.5	85.2	79.4	58.4
IoU Threshold 0.7	Part-A^2 Net	73.6	59.9	59.8	57.4	46	38.1	21.6
	PointPillars	**63.9**	**49.6**	**49.3**	**46**	**33.4**	**18.4**	**5.5**
	PointRCNN	73.3	46.8	47.3	46.9	43.9	36.4	20.8
	PV-RCNN	75.9	60.6	60.4	57.4	49	33.3	16.7
	SECOND	66.5	52.4	52.3	49.1	41.5	26.3	10.3
	Voxel R-CNN	84.6	67.9	67.9	64	54.1	37.1	16.4

TABLE VII: The LOAM averaged results of the error metrics: Position (m) and Rotation (degree).

	ATE_{pos}	ATE_{rot}	RE_{pos}	RE_{rot}
Original	0.316	0.57	0.389	0.82
4500×64 RI	0.321	0.2	0.387	0.83
4096×64 RI	0.313	0.21	0.39	0.84
2048×64 RI	0.294	0.17	0.388	0.82
1024×64 RI	0.394	0.17	0.388	0.82
512×64 RI	**0.610**	0.17	0.388	0.82
256×64 RI	**0.596**	0.2	0.387	0.82

is a 3D mapping technique running the odometry, point matching, and registration (mapping) algorithms simultaneously [67]. LOAM, and other SLAM algorithms that use different sensors, are widely used in various use cases, including autonomous vehicle, extended reality, and 3D reconstruction, and are one of the key perception tasks. We evaluate the quality impacts of point clouds reconstructed from different RI resolutions with A-LOAM [68] and the evaluator [71]. The experiments are with a sequence of 1101 LiDAR point clouds from the KITTI dataset. We show our evaluation results using two metrics: absolute trajectory error (ATE) and relative error (RE). While ATE calculates the root mean squared errors (RMSE) of position (ATE_{pos}) and rotation (ATE_{rot}) to the groundtruth, RE measures the relative relations of sub-trajectories in position (RE_{pos}) and rotation (RE_{rot}) [72].

Table VII shows the evaluation results of A-LOAM with different RIs. Based on the results, the LOAM algorithm works well even with high quantization and subsampling errors. Except for the increased ATE_{pos} for 512×64 and 256×64, other results are almost same with the result of the original data. Moreover, the LOAM paths of all cases are almost identical to each other as shown in Figure 11. After thorough analysis of the A-LOAM implementation, we find the mapping resolutions of A-LOAM are attributed to these results; the line and plane mapping resolutions of A-LOAM are 0.4 m and 0.8 m [68]. The increased ATE_{pos} for 512×64 and 256×64

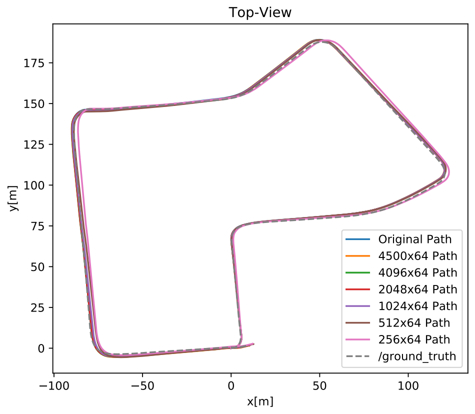


Fig. 11: The path results of LOAM with point clouds of different RIs.

are because the coarser subsampling causes loss of the sparse regions in the scene. Specifically, in Figure 6, the points over long distances are lost with coarser subsampling. The distance errors are reflected in ATE_{pos} because ATE calculates the RMSE over the whole path; there is no global reference in LOAM, and the early small errors can contribute to ATE more than the later errors [72]–[74]. For RE_{pos}, it calculates the averaged errors of separate sub-trajectories, and the distance errors are not accumulated over the whole path.

Summary. Based on the experiments, we demonstrate that FLiCR is suitable for enabling edge-assisted online perceptions to mobile users. Compared to the existing LiDAR point cloud compressions, it is *fast* in terms of the end-to-end compression/decompression latency, and *lightweight and efficient* in terms of energy usage and compression ratio. FLiCR achieves these benefits by affecting the quality of the point clouds using RI quantization and subsampling errors, and the end-to-end experiments of 3D object detection and LOAM show the impacts of the quality degradation on the downstream perception algorithms and their parameters. Even though the lossy RIs have a different effect on the perception performance based on each algorithm setting, ePSNR is able to quantify the point-wise and entropy-wise quality of a point cloud effectively. Thus, when optimizing the compression method, it would be crucial to co-design the compression system with awareness of the impact on downstream perceptions, and we leave this for future work.

VIII. Related Work

Given the popularity of 3D point clouds, there are many point cloud compression methods. Firstly, there are MPEG standard specifications: video-based point cloud compression (V-PCC) and geometry-based PCC (G-PCC) [31]. V-PCC converts 3D point clouds into 2D frames and compresses the frames with MPEG video codecs. G-PCC directly leverages the octree structure as the intermediate representation (IR), and compresses the octree of point clouds. Other than G-PCC, Google Draco [42] and Point Cloud Library (PCL) compressors [38] utilize tree structures including k-d tree and octree. After generating the tree structure from a point cloud, the occupancy information with the leaf nodes is coded, and entropy or arithmetic coding is applied to compress the coded information [39], [75]. For range image compression, Tu *et al.* present direct mapping of sensor data to 2D frames by each laser ID with precision and compress these raw RIs using image compression methods [26]. Other RI-based compression methods convert the raw sensor data from Cartesian coordinates into spherical coordinates by using the LiDAR sensor design [28], [29], [32]. Feng *et al.* propose spatial encoding in the plane granularity and temporal optimization with scene alignment and prediction by using IMU fusion. Even though these existing compression methods show decent compression performance, it is hard to apply them to our target use case of online remote perceptions because of their high latency magnitudes, as described in Section III.

Recently, there has been research to utilize machine learning (ML) for LiDAR point cloud compression. One popular approach is with the octree because the high compression ratio can be achieved by coding the tree into a more compact bytestream with well-predicted occupancy information of a given tree [75]. By fully utilizing the relationship of neighboring nodes in the octree, the state-of-the-art works train the ML models to predict the distribution of the octree nodes [30], [34], [40], [41]. With the predicted distribution, the occupancy information and nodes are effectively coded by assigning proper bits to each node of non-empty child nodes. For RI-based ML approaches, the spatial optimization is done by using the encoder and decoder networks trained with RIs of point clouds [27], [33]. Some of these ML algorithms achieve sufficiently low latency to run in real-time [30], [34]. However, they are not practical for mobile users, because they rely on high-end processors and GPUs, which are usually unavailable for mobile devices. Even if a mobile device has such computing resources, there is another issue with its limited battery.

IX. Limitations and Future Work

Although we show the effectiveness of FLiCR and ePSNR, there are still some remaining limitations. Firstly, as we observed with the end-to-end experiments, perception models pre-trained with the original data lose their predictive performance when used with point clouds reconstructed from lossy RIs. To alleviate this issue, there is an opportunity to make the perception models robust to point clouds from different RI resolutions. Another opportunity is to develop dedicated hardware logic for the processing steps in Figure 7. As shown in Figure 9, the RI conversion takes a large portion of the end-to-end latency. Accelerating the conversion process would further improve the latency benefits of FLiCR. In addition, ePSNR has a limitation. While ePSNR as a single-number metric effectively represents the point-wise and entropy-wise point cloud qualities, it requires two parameters: α and β. We manually set these parameters for our experiments, but it is not scalable. Therefore, there is a need to further develop a tuning methodology for these parameters, or to further refine the quality metric for LiDAR point clouds.

X. Conclusion

We describe the limitations of the existing point cloud compression methods for enabling LiDAR online perception on the edge. We propose a lightweight, low-latency, and efficient compression method by using RI and dictionary coding. For achieving the requirements, FLiCR fully leverages lossy RIs with quantization and subsampling. To quantify the quality loss by quantization and subsampling, we introduce a new metric, ePSNR, which reflects both the point-wise and entropy-wise qualities of a point cloud. We evaluate our compression method and demonstrate FLiCR is more appropriate for edge-assisted LiDAR online perceptions than the state-of-the-art compression algorithms. Compared to the existing algorithm most suitable for the target use case, FLiCR takes up to 80 percent less end-to-end latency while presenting $12\times$compression ratio. Our evaluation results with 3D object detection and LOAM show the impact of lossy RIs on the downstream perceptions and the effectiveness of ePSNR compared to the current quality metrics to capture this impact.

Acknowledgment

We would like to thank the anonymous reviewers. We are grateful to José Araújo, Héctor Caltenco, Bob Forsman, Per-Erik Brodin, and Gregoire Phillips for providing valuable feedback on this work and helping us improve its presentation. This work has been partially supported by NSF projects CCF-2217070 and CNS-1909769, the Applications Driving Architectures (ADA) Research Center, a JUMP Center co-sponsored by SRC and DARPA, and by funding and equipment gifts from VMware and Intel.

References

[1] Timothy Lee, "Lidar used to cost \$ 75,000—here's how apple brought it to the iphone," 2020.

[2] Velodyne Lidar, "Velabit: Velodyne's smallest lidar sensor," https://velodynelidar.com/products/velabit/, 2021.

[3] Intel, "Intel realsense™ lidar camera l515," https://www.intelrealsense.com/lidar-camera-l515/, 2021.

[4] Apple, "Apple unveils new ipad pro with breakthrough lidar scanner and brings trackpad support to ipados," 2020.

[5] M. Simon, K. Amende, A. Kraus, J. Honer, T. Samann, H. Kaulbersch, S. Milz, and H. Michael Gross, "Complexer-yolo: Real-time 3d object detection and tracking on semantic point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019, pp. 0–0.

[6] M. Simony, S. Milzy, K. Amendey, and H.-M. Gross, "Complex-yolo: An euler-region-proposal for real-time 3d object detection on point clouds," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, 2018, pp. 0–0.

[7] M. Ye, S. Xu, and T. Cao, "Hvnet: Hybrid voxel network for lidar based 3d object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 1631–1640.

[8] B. Yang, W. Luo, and R. Urtasun, "Pixor: Real-time 3d object detection from point clouds," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018, pp. 7652–7660.

[9] Y. Zhang, Z. Zhou, P. David, X. Yue, Z. Xi, B. Gong, and H. Foroosh, "Polarnet: An improved grid representation for online lidar point clouds semantic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 9601–9610.

[10] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Generation Computer Systems*, vol. 97, pp. 219–235, 2019.

[11] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, 2015, pp. 155–168.

[12] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–16.

[13] J. Heo, K. Bhardwaj, and A. Gavrilovska, "Poster: Enabling flexible edge-assisted xr," in *2021 IEEE/ACM Symposium on Edge Computing (SEC)*, 2021, pp. 465–467.

[14] I. E. Richardson, *H. 264 and MPEG-4 video compression: video coding for next-generation multimedia*. John Wiley & Sons, 2004.

[15] A. Grange, P. De Rivaz, and J. Hunt, "Vp9 bitstream & decoding process specification," *Version 0.6, March*, 2016.

[16] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *IEEE Transactions on circuits and systems for video technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[17] Intel, "Intel quick sync video, create, edit, and share video in a flash," 2011.

[18] Qualcomm, "Snapdragon 855+/860 mobile platform," 2021.

[19] Nvidia Corporation, "Nvidia video codec sdk," 2021.

[20] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[21] Y. Xiao, Y. Cui, P. Savolainen, M. Siekkinen, A. Wang, L. Yang, A. Ylä-Jääski, and S. Tarkoma, "Modeling energy consumption of data transmission over wi-fi," *IEEE Transactions on Mobile Computing*, vol. 13, no. 8, pp. 1760–1773, 2013.

[22] E. J. Vergara and S. Nadjm-Tehrani, "Energybox: a trace-driven tool for data transmission energy consumption studies," in *European Conference on Energy Efficiency in Large Scale Distributed Systems*. Springer, 2013, pp. 19–34.

[23] X. Zhang and K. G. Shin, "E-mili: Energy-minimizing idle listening in wireless networks," *IEEE Transactions on Mobile Computing*, vol. 11, no. 9, pp. 1441–1454, 2012.

[24] M. Li, Y.-X. Wang, and D. Ramanan, "Towards streaming perception," in *European Conference on Computer Vision*. Springer, 2020, pp. 473–488.

[25] K. Mammou, P. A. Chou, D. Flynn, M. Krivokuća, O. Nakagami, and T. Sugio, "G-pcc codec description v2," *ISO/IEC JTC1/SC29/WG11 N18189*, 2019.

[26] C. Tu, E. Takeuchi, C. Miyajima, and K. Takeda, "Compressing continuous point cloud data using image compression methods," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2016, pp. 1712–1719.

[27] C. Tu, E. Takeuchi, A. Carballo, and K. Takeda, "Point cloud compression for 3d lidar sensor using recurrent neural network with residual blocks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 3274–3280.

[28] J.-K. Ahn, K.-Y. Lee, J.-Y. Sim, and C.-S. Kim, "Large-scale 3d point cloud compression using adaptive radial distance prediction in hybrid coordinate domains," *IEEE Journal of Selected Topics in Signal Processing*, vol. 9, no. 3, pp. 422–434, 2014.

[29] H. Houshiar and A. Nüchter, "3d point cloud compression using conventional image compression for efficient data transmission," in *2015 XXV International Conference on Information, Communication and Automation Technologies (ICAT)*. IEEE, 2015, pp. 1–8.

[30] L. Huang, S. Wang, K. Wong, J. Liu, and R. Urtasun, "Octsqueeze: Octree-structured entropy model for lidar compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1313–1323.

[31] 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 Transactions on Signal and Information Processing*, vol. 9, 2020.

[32] Y. Feng, S. Liu, and Y. Zhu, "Real-time spatio-temporal lidar point cloud compression," in *2020 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2020, pp. 10 766–10 773.

[33] C. Tu, E. Takeuchi, A. Carballo, and K. Takeda, "Real-time streaming point cloud compression for 3d lidar sensor using u-net," *IEEE Access*, vol. 7, pp. 113 616–113 625, 2019.

[34] Z. Que, G. Lu, and D. Xu, "Voxelcontext-net: An octree based framework for point cloud compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 6042–6051.

[35] X. Sun, S. Wang, M. Wang, Z. Wang, and M. Liu, "A novel coding architecture for lidar point cloud sequence," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5637–5644, 2020.

[36] X. Sun, H. Ma, Y. Sun, and M. Liu, "A novel point cloud compression algorithm based on clustering," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2132–2139, 2019.

[37] F. Song, Y. Shao, W. Gao, H. Wang, and T. Li, "Layer-wise geometry aggregation framework for lossless lidar point cloud compression," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 12, pp. 4603–4616, 2021.

[38] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 1–4.

[39] O. Devillers and P.-M. Gandoin, "Geometric compression for interactive transmission," in *Proceedings Visualization 2000. VIS 2000 (Cat. No. 00CH37145)*. IEEE, 2000, pp. 319–326.

[40] S. Biswas, J. Liu, K. Wong, S. Wang, and R. Urtasun, "Muscle: Multi sweep compression of lidar using deep entropy models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 22 170–22 181, 2020.

[41] D. T. Nguyen, M. Quach, G. Valenzise, and P. Duhamel, "Multiscale deep context modeling for lossless point cloud geometry compression," in *2021 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*. IEEE, 2021, pp. 1–6.

[42] Google, "Draco: 3d data compression," 2018.

[43] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[44] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[45] M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan *et al.*, "Argoverse: 3d tracking and forecasting with rich maps," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8748–8757.

[46] J. Ziv and A. Lempel, “A universal algorithm for sequential data compression,” *IEEE Transactions on information theory*, vol. 23, no. 3, pp. 337–343, 1977.

[47] Z. C. Marton, R. B. Rusu, and M. Beetz, “On fast surface reconstruction methods for large and noisy point clouds,” in *2009 IEEE international conference on robotics and automation*. IEEE, 2009, pp. 3218–3223.

[48] L. Dagum and R. Menon, “Openmp: an industry standard api for shared-memory programming,” *IEEE computational science and engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[49] Velodyne Lidar, “Velodyne lidar hdl-64e,” 2018.

[50] M. Shevtsov, A. Soupikov, and A. Kapustin, “Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes,” in *Computer Graphics Forum*, vol. 26, no. 3. Wiley Online Library, 2007, pp. 395–404.

[51] D. Wehr and R. Radkowski, “Parallel kd-tree construction on the gpu with an adaptive split and sort strategy,” *International Journal of Parallel Programming*, vol. 46, no. 6, pp. 1139–1156, 2018.

[52] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha, “Fast bvh construction on gpus,” in *Computer Graphics Forum*, vol. 28, no. 2. Wiley Online Library, 2009, pp. 375–384.

[53] T. Karras, “Maximizing parallelism in the construction of bvhs, octrees, and k-d trees,” in *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics*, 2012, pp. 33–37.

[54] Z. Wu, F. Zhao, and X. Liu, “Sah kd-tree construction on gpu,” in *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, 2011, pp. 71–78.

[55] N. Salman, M. Yvinec, and Q. Mérigot, “Feature preserving mesh generation from 3d point clouds,” in *Computer graphics forum*, vol. 29, no. 5. Wiley Online Library, 2010, pp. 1623–1632.

[56] B. Guan, S. Lin, R. Wang, F. Zhou, X. Luo, and Y. Zheng, “Voxel-based quadrilateral mesh generation from point cloud,” *Multimedia Tools and Applications*, vol. 79, no. 29, pp. 20 561–20 578, 2020.

[57] M. W. Marcellin, M. J. Gormish, A. Bilgin, and M. P. Boliek, “An overview of jpeg-2000,” in *Proceedings DCC 2000. Data Compression Conference*. IEEE, 2000, pp. 523–541.

[58] FFmpeg team, “Ffmpeg, h.264 video encoding guide,” 2022.

[59] S. Shanmugasundaram and R. Lourdusamy, “A comparative study of text compression algorithms,” *International Journal of Wisdom Based Computing*, vol. 1, no. 3, pp. 68–76, 2011.

[60] O. D. Team, “Openpcdet: An open-source toolbox for 3d object detection from point clouds.”

[61] S. Shi, Z. Wang, J. Shi, X. Wang, and H. Li, “From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network,” *IEEE transactions on pattern analysis and machine intelligence*, 2020.

[62] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, “Pointpillars: Fast encoders for object detection from point clouds,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 697–12 705.

[63] S. Shi, X. Wang, and H. Li, “Pointrcnn: 3d object proposal generation and detection from point cloud,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 770–779.

[64] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li, “Pv-rcnn: Point-voxel feature set abstraction for 3d object detection,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 529–10 538.

[65] Y. Yan, Y. Mao, and B. Li, “Second: Sparsely embedded convolutional detection,” *Sensors*, vol. 18, no. 10, p. 3337, 2018.

[66] J. Deng, S. Shi, P. Li, W. Zhou, Y. Zhang, and H. Li, “Voxel r-cnn: Towards high performance voxel-based 3d object detection,” *arXiv preprint arXiv:2012.15712*, 2020.

[67] J. Zhang and S. Singh, “Loam: Lidar odometry and mapping in real-time.” in *Robotics: Science and Systems*, vol. 2, no. 9, 2014.

[68] C. S. Qin Tong, “Advanced implementation of loam.”

[69] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “Pointnet++: Deep hierarchical feature learning on point sets in a metric space,” *Advances in neural information processing systems*, vol. 30, 2017.

[70] B. Graham, M. Engelcke, and L. van der Maaten, “3d semantic segmentation with submanifold sparse convolutional networks,” *CVPR*, 2018.

[71] H. Gim, D. Cho, and J. Hong, “A framework for lidar slam algorithm evaluation.”

[72] Z. Zhang and D. Scaramuzza, “A tutorial on quantitative trajectory evaluation for visual (-inertial) odometry,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 7244–7251.

[73] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner, “On measuring the accuracy of slam algorithms,” *Autonomous Robots*, vol. 27, no. 4, pp. 387–407, 2009.

[74] W. Burgard, C. Stachniss, G. Grisetti, B. Steder, R. Kümmerle, C. Dornhege, M. Ruhnke, A. Kleiner, and J. D. Tardös, “A comparison of slam algorithms based on a graph of relations,” in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 2089–2095.

[75] R. Schnabel and R. Klein, “Octree-based point-cloud compression.” in *PBG@ SIGGRAPH*, 2006, pp. 111–120.