

# Been There, Scanned That: Nostalgia-Driven Point Cloud Compression for Self-Driving Cars

Ali Khalid  
Rochester Institute of Technology  
Rochester, NY, USA  
ak5013@rit.edu

Jaiaid Mobin  
Rochester Institute of Technology  
Rochester, NY, USA  
jm5071@rit.edu

Sumanth Rao Appala  
University of Pennsylvania  
Philadelphia, PA, USA  
appala@seas.upenn.edu

Avinash Maurya  
Rochester Institute of Technology  
Rochester, NY, USA  
am6429@rit.edu

Stephany Berrio Perez  
The University of Sydney  
Sydney, NSW, Australia  
stephany.berrioperez@sydney.edu.au

M. Mustafa Rafique  
Rochester Institute of Technology  
Rochester, USA  
mrafique@cs.rit.edu

Fawad Ahmad  
Rochester Institute of Technology  
Rochester, USA  
fawad@cs.rit.edu

## Abstract

An autonomous vehicle generates several terabytes of sensor data per day. A significant portion of this data consists of 3D point clouds produced by depth sensors such as LiDAR. This data is transferred to cloud storage, where it is utilized for training machine learning models or conducting analyses, e.g., forensic investigations in the event of an accident. To reduce network and storage costs, this paper introduces DejaView that searches for and uses redundancies on larger temporal scales (days and months) for more effective compression. We designed DejaView with the insight that the operating area of autonomous vehicles is limited and that vehicles mostly traverse the same routes daily. Consequently, the daily collected 3D data is likely similar to the data they've captured in the past. To capture this, the core of DejaView is a `diff` operation that compactly represents point clouds as delta w.r.t. 3D data from the past. Using two months of LiDAR data, DejaView can compress point clouds by a factor of 210 at a reconstruction error of only 15 cm.

## CCS Concepts

• **Computer systems organization** → *Embedded and cyber-physical systems*; • **Computing methodologies** → **Computer vision**; • **Information systems** → *Information storage systems*.

## Keywords

Autonomous Vehicles, 3D Point Cloud Compression, LiDAR

## ACM Reference Format:

Ali Khalid, Jaiaid Mobin, Sumanth Rao Appala, Avinash Maurya, Stephany Berrio Perez, M. Mustafa Rafique, and Fawad Ahmad. 2026. Been There, Scanned That: Nostalgia-Driven Point Cloud Compression for Self-Driving



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

*SenSys '26, Saint Malo, France*

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2309-4/26/05

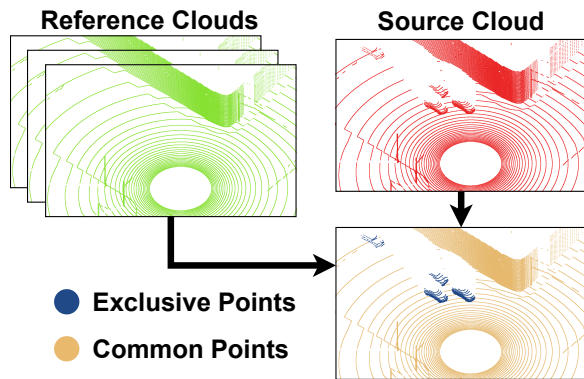
<https://doi.org/10.1145/3774906.3800467>

Cars. In *ACM/IEEE International Conference on Embedded Artificial Intelligence and Sensing Systems (SenSys '26)*, May 11–14, 2026, Saint Malo, France. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3774906.3800467>

## 1 Introduction

Autonomous vehicles (AVs) rely on 3D sensors such as LiDARs, cameras, and Radars to understand their surroundings. Beyond real-time operation, the data generated from these sensors is used offline for forensic analysis [17, 27, 50, 51] to verify safety compliance and investigate insurance disputes. This data is also used to train and evaluate machine learning models for perception, prediction, and planning [19, 37, 39, 65]. As such, the sensor data generated from the AVs is *uploaded offline, after the vehicle is parked, to cloud storage for long-term retention* [34]. However, an AV's sensor suite can generate several terabytes (TBs) of data per day [27, 63]. Among these sensors, LiDAR, which generates 3D point clouds containing 3D points defined by their spatial coordinates and other attributes, is the most data-intensive sensor. A 128-beam LiDAR produces over 5.2 million 3D points per second (equivalent to a data rate of 1 Gbps). Moreover, multiple LiDARs per vehicle (e.g., 4 LiDARs on Waymo's 6th-generation driver system [32]) can produce 10s of terabytes of data per day. As the deployment of AVs expands, the sheer volume of sensor data will strain both network resources (for offline data transfer) and cloud infrastructure (for storage).

In this paper, we build a framework, called DejaView, for compressing 3D point clouds generated by LiDARs to reduce network transmission and cloud storage costs. Once the vehicle is parked after its daily operation, DejaView compresses 3D point clouds collected by the vehicle throughout the day and then transfers the compressed data to the cloud for storage. Although generic point-cloud compression techniques (e.g., Draco [24], Octree [48], and MPEG GPCC [38]) exist, we posit that the characteristics of AV data present an opportunity for significantly higher compression ratios. This opportunity stems from two observations: a) AVs, like human drivers, often traverse similar routes (e.g., the daily commute to work), and b) AVs are mostly geofenced and their operation area is limited [4]. Consequently, *most of the data captured on any given*



**Figure 1:** The source cloud can be reconstructed using the common and exclusive points w.r.t. the reference cloud.

day resembles data collected in the past days, weeks, or months, particularly for static environmental elements, such as road structures, buildings, and traffic signs. This redundancy across larger temporal scales presents a promising avenue to improve compression for AV-generated LiDAR data.

Existing compression techniques, such as those used in video streaming [29, 35, 36, 66], typically focus on inter-frame compression over short timescales. These techniques store every subsequent frame as a delta (or difference) with respect to the previous frame. Although effective in capturing frame-to-frame redundancies, these methods fail to take advantage of similarities across days, weeks, or months when vehicles operate in the same areas. Therefore, the overarching challenge in this paper is to identify redundancies across larger temporal scales and use them to compress 3D data.

This paper addresses the above challenge by using spatial hints in AVs. Specifically, we make use of the fact that AVs are equipped with GPS and 3D maps, allowing them to position themselves precisely in the world. Consequently, for any point cloud we want to compress (Fig. 1: source cloud in red), we use the AV’s position to retrieve the historically proximate point clouds (Fig. 1: reference clouds in green). Then, we compare the source cloud with the reference clouds to identify two sets of points: a) *common points* (Fig. 1: brown points) that exist in both point clouds, and b) *exclusive points* (Fig. 1: blue points) that exist in source cloud but not in the reference clouds. In our proposed framework, we compactly store the source cloud using only the 3D positions of the *exclusive points* and references (or pointers) to the *common points*. This compressed version is sent over the network and stored in the cloud. Although conceptually simple, implementing this presents significant challenges.

**Challenges.** The core of our proposed solution is accurately computing the difference between the source cloud and the reference cloud. This computation embodies a trade-off between compression ratio, reconstruction error, and latency. Reconstruction error is the difference between an original point cloud and its reconstructed version after compression.

What to compute the difference against? Compression performance depends on the number of exclusive points in the source cloud. Fewer exclusive points lead to higher compression ratios. Compared to a single reference cloud, if we compute the difference against multiple consecutive reference clouds, the source cloud will share more common points with the reference clouds and hence

fewer exclusive points. However, comparing against multiple reference clouds can incur significant latency due to the increased number of point comparisons. Moreover, this inadvertently reduces compression ratios, as we show in §2.

How to compute the difference? To compute the difference between point clouds, we use the nearest neighbor search to find, for each point in the source cloud, if an identical point exists in the reference cloud, and vice versa. For this, coarse-grained region-based approaches (e.g., Octree [48]) are fast, but could misclassify identical points if grouped into different regions. This leads to sub-optimal compression. Conversely, fine-grained point-wise approaches (e.g., k-d tree [11]) are more precise but incur significant latency.

**Contributions.** To address the above challenges, this paper makes the following contributions:

- We propose a novel technique for compressing point clouds by leveraging redundancies in point clouds across temporal scales using their spatial relationships.
- We demonstrate the efficacy of this approach by building an end-to-end system, *DejaView*, to compress AV LiDAR point clouds.
- *DejaView* proposes a cascaded difference computation algorithm that uses a single source cloud and a collection of reference clouds to achieve high compression without trading off latency.
- We develop an accelerated nearest neighbor search algorithm that uses coarse-grained and fine-grained searches to enable fast compression without trading off reconstruction quality.

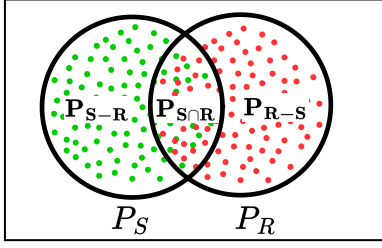
On a real-world vehicle LiDAR dataset collected over 2 months, consisting of 297K point clouds, an end-to-end implementation of *DejaView* achieves a compression ratio of 210 with a reconstruction error of less than 15 cm, significantly outperforming prior works.

## 2 Motivation and Background

**The Need for Storing Sensor Data.** Beyond real-time perception, the sensor data generated by AVs is useful for several offline applications, including (but not limited to) forensic analysis, training, and verification of machine learning models. For example, this data can be used to reconstruct scenarios leading up to events such as traffic accidents [53], unexpected AV behavior, corner cases, system failures, and even cyberattacks [51]. 3D reconstructions of these events can enable comprehensive forensic analysis, resolve insurance disputes, and, most importantly, enhance root cause identification.

Beyond forensic applications, large-scale real-world datasets collected from AVs are critical for training, refining, and evaluating machine learning models across the autonomous driving stack, including perception, planning, and control modules [19, 37, 39, 65]. By integrating this data into simulation frameworks such as Waymo’s Simulation City [64], developers can safely reproduce diverse traffic conditions, generate statistically representative edge cases, and continuously improve model robustness without requiring extensive on-road testing. These real-world-grounded simulations allow ML models to adapt to evolving urban layouts, new mobility patterns, and rare driving events, ultimately accelerating the safe and scalable deployment of autonomous driving systems.

**The Volume of AV LiDAR Data.** With multiple LiDARs mounted on an AV (e.g., four LiDARs on Waymo’s 6th-generation driver system [32]), just ten hours of operation, even with a conservative estimate, can produce 12 TBs of LiDAR data daily, 360 TB monthly,



**Figure 2:** Common and exclusive points in two point clouds.

and 4 PB annually for a single AV. Using standard cloud storage pricing [2], the monthly storage cost for a single AV’s data would exceed \$7,000. For a fleet of just 100 AVs, the annual storage costs would reach well over \$9 million. Although these data are uploaded to cloud storage offline when the AV is parked, the sheer volume of data can lead to an unmanageable data transfer backlog. Even with a high-end WiFi 6 connection that achieves sustained speeds of 1.2 Gbps [8], uploading a single day of data would take nearly 40 hours, i.e., four times the amount of time to collect the data. Therefore, AV LiDAR data must be compressed to minimize both network transmission delays and long-term cloud storage costs.

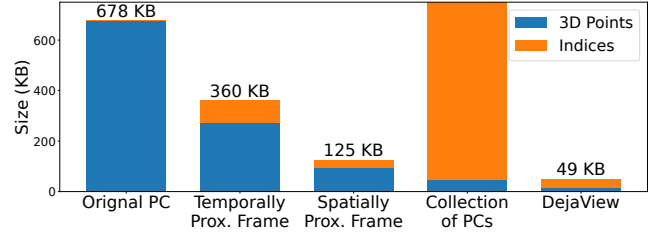
**3D Point Clouds.** Depth perception sensors, such as LiDARs, generate collections of points in which each point is defined by its 3D position  $(x, y, z)$  and possibly other attributes such as color and intensity. Point clouds are data-intensive and can be large in size. A 128-beam LiDAR can produce more than 2.6 million 3D points per second, which is equivalent to a data rate of 1 Gbps.

**Point Cloud Compression Techniques.** Existing point cloud compression techniques, e.g., Octree [33] and Draco [24], leverage spatial redundancy within a point cloud to achieve compression. In octree-based methods, the point cloud is recursively subdivided into smaller volumetric units (voxels) to organize points hierarchically, which helps in reducing spatial redundancy. This octree representation helps in compressing point clouds by enabling efficient encoding where only subdivisions with points are stored. Additionally, octree-based methods use interframe redundancies to achieve a better compression ratio. Draco, on the other hand, uses a k-d tree to reorder points to put spatially close points together. This reordering helps Draco improve the efficiency of entropy encoding. Draco also uses quantization as a key technique to improve compression for point clouds.

**Background Subtraction.** Background subtraction, which we refer to as a *diff* operation, finds the exclusive points in  $P_S$  w.r.t.  $P_R$  [33]. The operation *diff* determines whether, for each point  $s$  in  $P_S$ , there is an identical point  $r$  in  $P_R$ . To implement this, for each point  $s$  in  $P_S$ , we find the nearest neighboring point  $r$  in  $P_R$  (Equation 1). If this point  $r$  is within the distance range  $d$  (we call this *distance threshold*) from  $s$ , we consider  $s$  and  $r$  to be common or identical points ( $P_{S \cap R}$  in Fig. 2). If not, then  $s$  is a unique or exclusive point in  $P_S$  w.r.t.  $P_R$ .

$$\text{diff}(P_S, P_R) = \{s \in P_S \mid \min_{r \in P_R} \|s - r\|_2 > d\} \quad (1)$$

We have two sets of points from the *diff* operation: a) points common to  $P_S$  and  $P_R$  ( $P_{S \cap R}$ ), and b) exclusive points in  $P_S$  w.r.t.  $P_R$  ( $P_{S-R}$  in Fig. 2). To find exclusive points in  $P_R$  w.r.t.  $P_S$  ( $P_{R-S}$



**Figure 3:** Compression performance for strawman pipelines.

in Fig. 2), the positions of  $P_S$  and  $P_R$  would be swapped in Equation 1. In this paper, the *diff* operation is used for point-cloud compression.

**Point Cloud Compression using *diff*.** To compress the source cloud ( $P_S$ ) using a reference cloud ( $P_R$ ), we assume that  $P_R$  is present for both compression and decompression. In the first step, we perform a *diff* operation of  $P_S$  w.r.t.  $P_R$  i.e.,  $\text{diff}(P_S, P_R)$ . From this operation, we obtain: a)  $P_{S-R}$ , the exclusive points in  $P_S$  w.r.t.  $P_R$  and b)  $P_{S \cap R}$ , the common points between  $P_S$  and  $P_R$ . Next, we perform a reverse *diff* operation i.e.,  $\text{diff}(P_R, P_S)$  to determine the exclusive points  $P_{R-S}$  in  $P_R$  w.r.t.  $P_S$ . Using only  $P_R$  and the two sets of exclusive points ( $P_{S-R}$  and  $P_{R-S}$ ), we can reconstruct  $P_S$  as shown below (Equation 2):

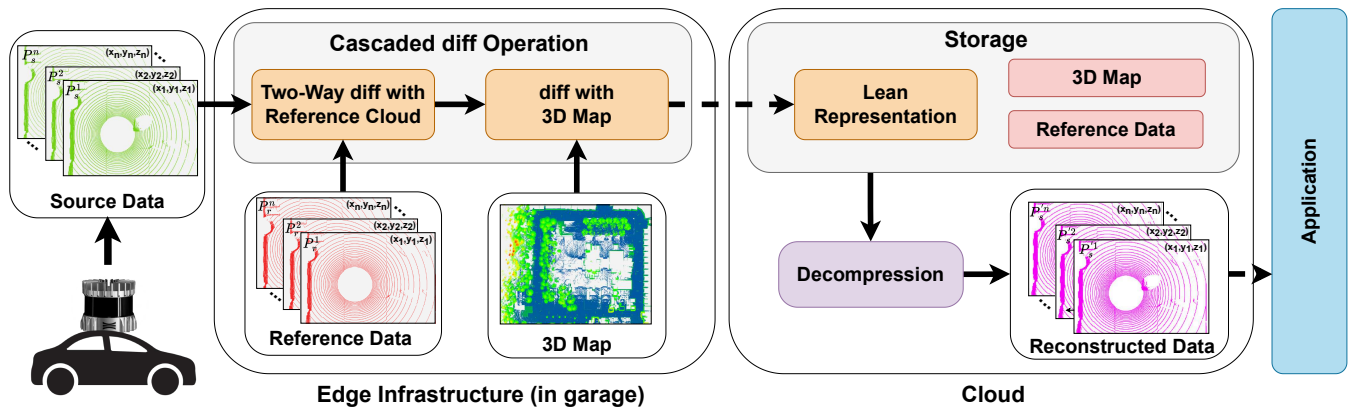
$$P_S = P_R - P_{R-S} + P_{S-R} \quad (2)$$

Of the three sets of points that need to reconstruct  $P_S$ , only the 3D positions of the points exclusive to source cloud ( $P_{S-R}$ ) is needed as reference cloud is already present at the end of the decompression and so are the points exclusive to it ( $P_{R-S}$ ). To retrieve the exclusive points ( $P_{R-S}$ ) from reference cloud, we only need their indices. Using the 3D positions of exclusive points in the source cloud ( $P_{S-R}$ ) and indices of  $P_{R-S}$ , we can reconstruct  $P_S$  (Equation 2).

**Strawman Pipelines for Compression.** To motivate DejaView, we build three strawman pipelines that use the *diff* operation to compress point clouds and compare their performance (Fig. 3). We used 10 days of data generated from CARLA [12], each with 500 point clouds, to evaluate these pipelines. All pipelines use the same *distance threshold*, i.e., 10 cm in this experiment.

**Temporally Proximate Frame.** This approach adapts a technique similar to video streaming, i.e., representing source cloud as a difference w.r.t. to a point cloud from the previous frame. Compared to the raw point cloud (first bar in Fig. 3), this approach reduces the size of the data by 1.9x, on average. For each bar, the blue area represents the size of the 3D points exclusive to source cloud, while the orange area represents the size of the indices for points that are exclusive to the reference cloud (in this case, the previous frame).

**Spatially Proximate Frame.** The second pipeline uses DejaView’s idea i.e., compress the source cloud using a spatially proximate reference cloud from the past. This reduces the size of the point cloud by 5.4x (third bar in Fig. 3), on average. This is because source cloud shares more 3D points with a spatially proximate reference cloud (as opposed to a temporally proximate one). This has a two-fold effect. First, there are fewer exclusive points in the source cloud to store 3D positions for and fewer exclusive points in the reference cloud to store indices for. DejaView’s *diff* operation can reduce the point cloud size by



**Figure 4:** Overview of DejaView. The AV generates source data, which is compressed at edge infrastructure (e.g., at garage or parking lot) to produce a lean representation. This lean representation is transmitted over the network to the cloud for storage. When requested by an application, the cloud decompresses the lean representation to reconstruct the data and delivers it to the application.

14x, followed by a series of techniques (§3) to further increase the compression ratio to 210x.

A Collection of Point Clouds. Intuitively, using a collection of reference clouds from the past should result in improved compression, because there is a higher probability of finding more common points and therefore fewer exclusive points for source cloud. However, this is not true, as shown by the fourth bar in Fig. 3. In this experiment, we used 2000 reference clouds to find exclusive points in source cloud. Although this reduces the number of exclusive points in source cloud, it significantly increases the number of exclusive points to store for reference clouds. This is because now every reference cloud will have a separate set of exclusive points w.r.t. the source cloud.

As a result, the overhead of storing their indices is many times larger than the size of exclusive points of the source cloud. Moreover, the size of the set of indices increases with the number of reference clouds. So, *this approach incurs four orders of magnitude higher latency* relative to comparing against a single-point cloud.

**Challenges.** Fig. 3 shows the effectiveness of DejaView’s proposed approach, achieving a 3.1x compression ratio. Although intuitively using more reference clouds should improve compression, our experiments show that this is not true and can be computationally expensive. The core challenge lies in finding the right balance: DejaView must identify more commonalities between the source cloud and reference clouds without introducing excessive overhead in managing these relationships. Furthermore, DejaView’s approach must find a balance between reconstruction accuracy and latency.

### 3 DejaView Design

**Overview.** To illustrate how DejaView operates, consider an AV that collects a sequence of point clouds ( $P_S^1, P_S^2, \dots, P_S^n$ ) on a given day. The online perception module directly processes these point clouds for localization and scene understanding (object detection, tracking). The AV also temporarily stores these point clouds in the high-speed on-board memory before uploading them to the cloud for long-term storage. We refer to these point clouds as source clouds (Fig. 4). DejaView compresses source clouds to build a compact representation for each source cloud when the AV is parked

in the garage at the end of its operation. These compact representations of source clouds are sent to the cloud for storage. DejaView decompresses them if they are needed for other applications (e.g., training, testing, or forensic analysis).

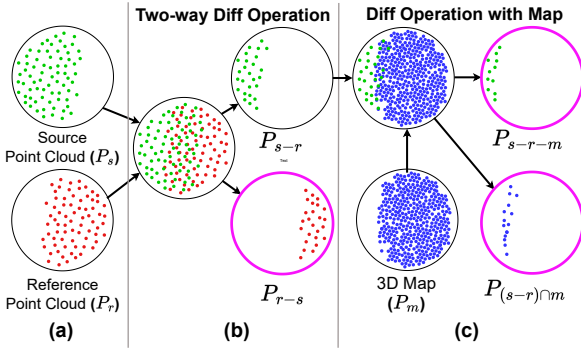
At its core, DejaView reduces the required network bandwidth and storage footprint by minimizing the number of points to store for a given source cloud. To do this, DejaView uses a reference dataset that contains reference clouds  $\{P_R^1, P_R^2, \dots, P_R^n\}$  (§3.3). For each given source cloud, DejaView finds the approximate reference cloud from the reference dataset. With a cascaded diff operation (§3.1), DejaView computes the difference between source cloud, its closest reference cloud, and the on-board 3D map<sup>1</sup>.

DejaView reconstructs source clouds on demand in the cloud upon application requests and transmits them accordingly. It reconstructs source clouds using their corresponding reference clouds in the reference dataset and the 3D map, as shown in Fig. 4. We assume that the 3D map is available onboard the AV, a common practice in the AV industry [40, 56], as it is essential for navigation and operation. The reference dataset (§3.3) is stored on edge infrastructure, e.g., in a garage where the AV is parked during compression, to reduce pressure on the AV’s onboard storage. DejaView also assumes that both the 3D map and reference dataset are available in the cloud during decompression. To enable this, the AV uploads a copy of the 3D map and reference dataset to the cloud once during its lifetime. Maintaining these datasets on AV and cloud reduces network load during compression and decompression phases.

#### 3.1 Cascaded diff Operation

DejaView uses a cascaded diff operation to compress source clouds using reference clouds and a 3D map. This consists of: (a) a two-way operation diff that compresses source cloud w.r.t. a reference cloud (Fig. 5b), followed by (b) a single-way diff operation that compresses the exclusive points of source cloud w.r.t. a 3D map (Fig. 5c). In the second operation, *instead of storing the exclusive points for the 3D map, DejaView stores the common points*. This

<sup>1</sup>A dense 3D point cloud used by the AV for localization within their environment



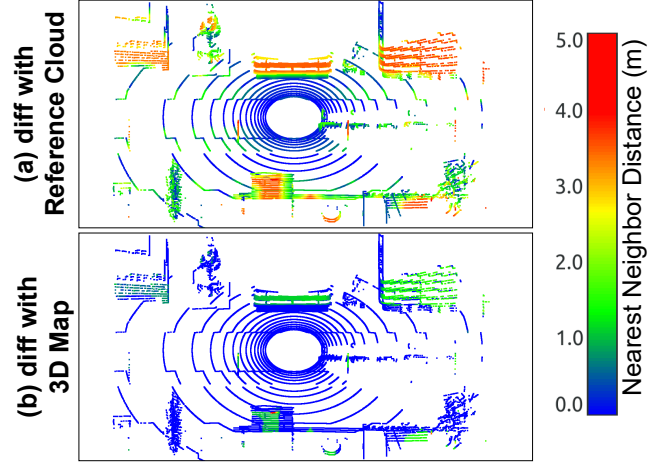
**Figure 5:** DejaView’s cascaded diff operation uses a three step process to compute a compact representation for the source cloud using a reference cloud and a 3D map. The point clouds with purple outlines constitute the compact representation.

significantly reduces the exclusive points for source cloud without the additional overhead of the indices for reference clouds.

**diff with a Reference Cloud.** In the first stage of this operation, DejaView retrieves a reference cloud ( $P_R$ ) from the reference dataset for a given source cloud ( $P_S$ ). The reference cloud is the closest point cloud in the 3D space to the source cloud (§3.3 describes the mechanism of selecting reference cloud). Next, DejaView performs a two-way diff operation between  $P_S$  and  $P_R$ . The first diff operation determines the exclusive points ( $P_{S-R}$ ) in source cloud w.r.t. and reference cloud (Fig. 5b). The second diff operation determines the exclusive points ( $P_{R-S}$ ) in the reference cloud w.r.t. the source cloud (Fig. 5b). As a result of this two-way diff operation, the compact version of source cloud contains: a) 3D positions of exclusive points of source cloud ( $P_{S-R}$ ), b) a pointer to reference cloud, and c) indices of the exclusive points of reference cloud ( $P_{R-S}$ ). However, as shown in §2, this operation only yields a compression ratio of 5.4x.

The point density (points per volumetric unit) of LiDAR point clouds (Fig. 6) differs greatly throughout the point cloud. Regions near the sensor are denser than those farther away due to the radial positioning of laser beams in a LiDAR’s mechanical enclosure. Fig. 6a shows how this non-uniform distribution of points affects a diff operation between source cloud and reference cloud i.e., diff ( $P_S, P_R$ ). For a source cloud captured from AV, we color-code every point to show the distance to its nearest neighbor in reference cloud. The blue points, in the denser regions near the sensor, are those for which source cloud finds an identical point in reference cloud. Points in regions farther away (green, yellow, and red) are those for which we cannot find neighboring points close by, and hence are classified as exclusive points.

**diff with the 3D map.** To improve compression, DejaView further reduces the number of exclusive points to store using a collection of point clouds or a 3D map ( $P_M$ ). However, as discussed in §2 and shown in Fig. 3, this is undesirable for two reasons. This operation can adversely affect the compression ratio because the set of exclusive points for the 3D map w.r.t. to the source cloud ( $P_{M-S}$ ) can be significantly large. Moreover, the second diff operation (i.e., diff ( $P_M, P_S$ )) of the two-way diff operation between the 3D map and the source cloud can incur significant latency because DejaView would need to find the nearest neighbors for all points



**Figure 6:** diff operation with a 3D map (b) can significantly reduce the number of points to store (non-blue points) as compared to a diff against a reference cloud (a), which is effective only for points closer to the LiDAR sensor.

in the 3D map. To address these challenges, *DejaView* uses an intelligently designed diff operation that ensures both low latency and high compression ratio.

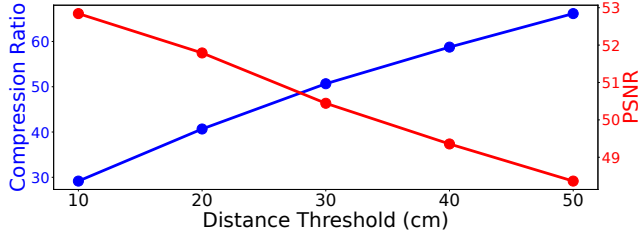
To optimize both the compression ratio and the latency, *DejaView* makes two careful design decisions. Firstly, *DejaView* only uses the set of exclusive points in source cloud ( $P_{S-R}$ ) for the diff operation. Secondly, it modifies the diff operation to get the common points between the two input point clouds and uses them in the compact representation. Putting these together, *DejaView* performs only a one-way diff operation using source cloud’s exclusive points ( $P_{S-R}$ ) against AV’s on-board 3D map ( $P_M$ ), i.e., diff ( $P_{S-R}, P_M$ ). This operation yields two sets of points (Fig. 5c): a) points exclusive to the source cloud w.r.t. both the reference cloud and the 3D map ( $P_{S-R-M}$ ), and b) common points between source cloud and the 3D map that do not exist in the reference cloud ( $P_{(S-R) \cap M}$ ). Using these sets of points, we can reconstruct  $P_{S-R}$  (Equation 3).

$$P_{S-R} = P_{S-R-M} + P_{(S-R) \cap M} \quad (3)$$

This operation has multiple benefits. First, performing a diff using exclusive points in the source cloud (i.e., diff ( $P_{S-R}, P_M$ )), instead of using the entire source cloud (i.e., diff ( $P_S, P_M$ )) significantly reduces latency. Second, because we can reconstruct  $P_{S-R}$  using  $P_{S-R-M}$  and  $P_{(S-R) \cap M}$  (Equation 3), the second more compute-intensive diff operation (i.e., diff ( $P_M, P_{S-R}$ )) can be avoided. Third, it improves the compression ratio by replacing a large number of exclusive points in the map with common points between the 3D map and exclusive points in source cloud ( $P_{(S-R) \cap M}$ ).

The compact representation of *DejaView* of a source cloud consists of: a) 3D points exclusive to source cloud w.r.t. the reference cloud and a 3D map ( $P_{S-R-M}$ ), b) reference to the reference cloud and indices of points exclusive to it w.r.t. to the source cloud ( $P_{R-S}$ ), and c) indices of points in the 3D map that are common with exclusive points in source cloud w.r.t. to reference cloud  $P_{(S-R) \cap M}$ .

**Further Compression.** Although *DejaView* stores indices as opposed to 3D positions of exclusive points in reference cloud and common points with the 3D map, these sets of indices contain large integer values, primarily because of the large numbers of points in



**Figure 7:** As the distance threshold ( $d$ ) increases, compression ratio increases at the cost of PSNR (i.e., reconstruction error increases).

the point clouds. This can adversely affect compression. To address this, DejaView uses delta encoding [55] to convert large integer values to smaller values. First, DejaView sorts the indices in descending order. Then, it applies delta encoding, which stores every subsequent index as a function of the previous index (Equation 4).

$$DE(I) = \{i_1, \Delta i_2, \Delta i_3, \dots, \Delta i_n\} \quad (4)$$

$$\text{where } \Delta i_x = i_{x-1} - i_x$$

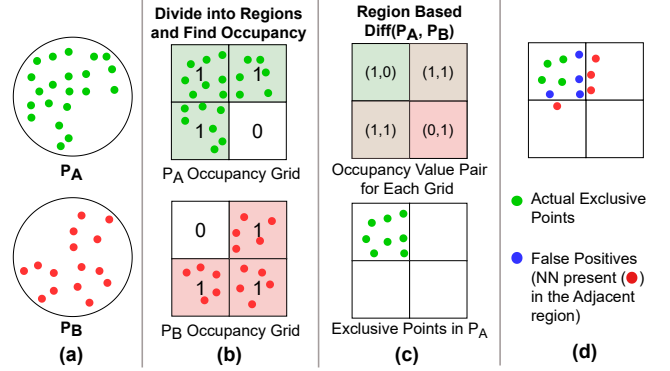
To improve compression, DejaView uses a hybrid technique, i.e., Draco [24] and LZMA [42] for compressing 3D points ( $P_{S-R-M}$ ) and indices, respectively. DejaView concatenates the two sets of delta-encoded indices ( $P_{R-S}$  and  $P_{(S-R) \cap M}$ ). It also appends their total counts to this list to separate the two at decompression time. It applies LZMA compression [42] to this list. After that, it compresses 3D points using Draco and concatenates the LZMA and Draco outputs. It also prepends the Draco output size (in bytes) to the final output to separate the Draco and LZMA parts during decoding. The compressed version of source cloud is sent over the network and stored in the cloud.

**Decompression.** When source cloud is needed for processing, DejaView uses the first four bytes of the compressed source cloud to separate the Draco and LZMA data. Next, it decompresses the Draco output to retrieve the set of 3D points ( $P_{S-R-M}$ ). Next, it decompresses the LZMA stream. From this, it separates the two sets of indices ( $P_{R-S}$  and  $P_{(S-R) \cap M}$ ) using the first integer in the decompressed output. Next, it uses delta decoding to retrieve the original set of indices. Finally, it retrieves reference cloud and the 3D map and reconstructs the source cloud (Equation 5).

$$P_S = P_R + P_{(S-R) \cap M} + P_{S-R-M} - P_{R-S} \quad (5)$$

**Compression Ratio Vs. Reconstruction Error.** For points in source cloud and reference cloud to be identical, they must be within the *distance threshold*  $d$  from each other (Equation 1). This knob in DejaView controls the amount of compression we apply to source cloud. If the *distance threshold* is high, more points in source cloud will be classified as common points and fewer will be exclusive points. Consequently, the compression ratio will increase (Fig. 7). However, this comes at the cost of the Peak Signal-to-Noise Ratio (PSNR), a proxy that we use for the reconstruction error (defined in §4). A higher PSNR shows that the reconstructed source cloud is similar to the raw source cloud. As the compression ratio increases because of the *distance threshold*, the PSNR decreases.

For example, the point  $a$  in source cloud is at  $(0,0,0)$ , whereas its nearest neighboring point  $b$  in reference cloud is at  $(0,0,0.4)$ . If the *distance threshold* is 0.5 m, then  $a$  and  $b$  would be classified as



**Figure 8:** Region-based diff operations tend to aggressively classify exclusive points, often resulting in false positives.

identical points. Thus, we would not need to store the 3D position of  $a$ , thus improving the compression ratio. At reconstruction time,  $a$  would be assigned the same position as  $b$  i.e.,  $(0,0,0.4)$ , which is 40 cm from its original position. As a result, the PSNR will be low. Conversely, if the *distance threshold* was smaller, i.e., 0.1 m, the  $a$  would be an exclusive point. So, we would need to store the 3D position of  $a$ , thereby reducing the compression ratio. At reconstruction time,  $a$  would be assigned its original position, i.e.,  $(0,0,0)$ , thus improving PSNR.

### 3.2 Efficient diff Operation

**The Problem.** To compress every source cloud, DejaView must perform three diff operations, specifically, two with a reference cloud and one with a 3D map. These diff operations must be fast to guarantee same-day compression and upload. If not, this can cause a backlog in cloud transfer and storage (§2).

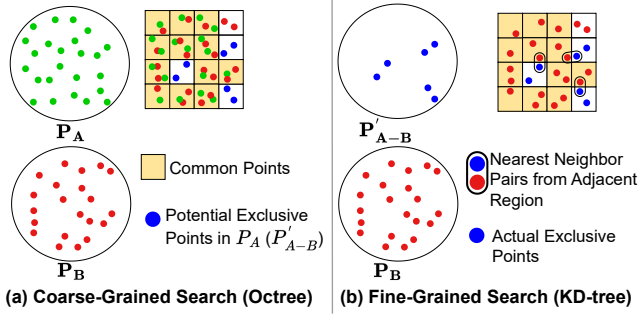
**Point-based Techniques.** Core to the diff operation is the computation of the nearest neighboring point. In general, there are two techniques to determine the nearest neighbor. Point-based approaches iterate through all points in one point cloud and determine their nearest neighboring point in the other cloud. These techniques use data structures, for example, the k-d tree [11], to speed up the nearest-neighbor search.

**Region-based Techniques.** Region-based techniques (Octree [33], Voxel Grid [15, 72]) partition point clouds into multiple regions (Fig. 8b). Each region has a flag that represents occupancy. The flag is set if there are points within the region (Fig. 8b). If a region in the first point cloud is occupied but the corresponding region in the second point cloud is not, then all points in the first point cloud's region are classified as exclusive. As such, in Fig. 8c, the top-left region of  $P_A$  is classified as exclusive w.r.t.  $P_B$ , and the bottom-right region of  $P_B$  is classified as exclusive w.r.t.  $P_A$ . However, a point's nearest neighbor may not be in the same region. For example, multiple points in the top-left region of  $P_A$  have nearest neighbors in the top-right region of  $P_B$ . Because region-based techniques do not consider this, these points are misclassified as exclusive points (blue points in Fig. 8d).

**Quantitative Comparison.** To compare the two approaches, we used them to perform a two-way operation diff between multiple source clouds and reference clouds. For this experiment, we measured the compute latency and compression ratio for these

diff Technique	Latency (ms)	Compression Ratio
Point-based (K-d tree [11])	663	3.4
Region-based (Octree [33])	145	2.3
DejaView	405	3.4

**Table 1:** Point-based approaches improve compression (at the cost of latency), whereas region-based techniques are faster (at the cost of compression). DejaView’s hybrid approach is both fast and improves compression.



**Figure 9:** DejaView uses a hybrid technique that uses coarse-grained and fine-grained searches to ensure low latency and high compression.

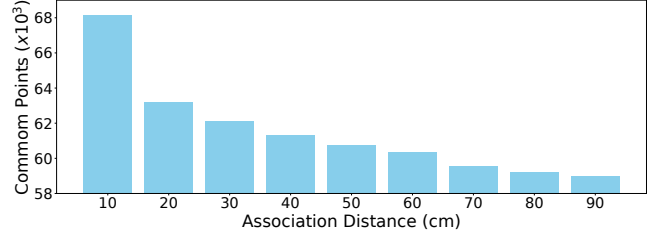
operations (Tbl. 1). Point-based techniques have higher compression ratios but can be slow. Region-based techniques are faster but have lower compression because they tend to have false positive exclusive points (as explained above). As a result, region-based techniques overestimate the number of exclusive points, leading to a decrease in compression.

**Our Approach.** To address these challenges, DejaView uses a hybrid search in its diff operations. This consists of a coarse-grained search followed by a fine-grained search. Assume DejaView performs a diff operation between  $P_A$  and  $P_B$ . To do this, DejaView loads both point clouds into an octree and performs a region-based search. This operation quickly identifies the *potential* exclusive points ( $P'_{A-B}$ ) between two points clouds (Fig. 9a). Knowing that this can produce false positive results, DejaView refines this with a fine-grained search. For this, DejaView loads  $P_B$  onto a k-d tree. Then, for each *potentially* exclusive point ( $P'_{A-B}$ ), it queries the k-d tree for its nearest neighbor in  $P_B$  (Fig. 9b). Once it retrieves the nearest neighbor of the point, it classifies it as an *actual* exclusive point if it does not lie within the predefined *distance threshold*. Otherwise, it is classified as an identical or common point. In this way, DejaView can remove false exclusive points.

DejaView’s hybrid diff operation is both fast and ensures high compression (Tbl. 1). The coarse-grained search ensures high computational efficiency. The fine-grained search mitigates their propensity to overestimate exclusive points. Moreover, since the fine-grained search is used for fewer points, it reduces the overall latency overhead. Populating a k-d tree can be computationally expensive. Because all source clouds perform a diff against the same 3D map, DejaView preloads the 3D map into a k-d tree representation offline and reuses this for every frame.

### 3.3 Reference Dataset

DejaView uses a reference dataset consisting of reference clouds  $\{P_R^1, P_R^2, \dots, P_R^n\}$  to compress and decompress source clouds. This data set consists of point clouds that AVs have collected in the past



**Figure 10:** Close-by source cloud and reference cloud pairs (lower association distances) tend to have more common points and fewer exclusive points, hence leading to better compression.

for a given region on any given day. We assume that this data set is stored in uncompressed format in the edge compute (e.g., the parking garage) and in the cloud. AVs within a similar geographic region (e.g., town or city) can share this data set. Each reference cloud has a pose (3D position and rotation) that describes where the point cloud was captured.

Selecting reference cloud for a given source cloud is crucial to maximize the compression ratio. Ideally, if the reference cloud points are in the same 3D positions as those in source cloud, this would yield maximum compression. However, determining point-level similarity can be computationally expensive. Instead, DejaView uses spatial proximity as a proxy for point-level similarity. This is based on the intuition that the static parts of the environment will be structurally similar across larger temporal scales.

For quantitative evaluation, we used 2000 pairs of source cloud and reference cloud and measured the number of common points by changing the 3D distance (association distance) between them. As Fig. 10 shows, if source cloud and reference cloud are close together, they have a larger number of common points between them, leading to higher compression ratios. We observe a sharp decrease in common points if the distance between source cloud and reference cloud is greater than 10 cm. These 3D positions for source cloud and reference cloud were already computed when AVs collect these point clouds during their normal operation. To reduce latency, DejaView performs this association using vectorized calculations. The 3D positions of source clouds and reference clouds are loaded in separate matrices to compute the Euclidean distances for all pairs of source cloud and reference cloud. The reference cloud with the lowest distance from source cloud is associated with it.

## 4 Performance Evaluation

### 4.1 Evaluation Methodology

**Implementation.** We have implemented DejaView as three main modules using C++ and Python. The compression and decompression modules, that use Point Cloud Library (PCL [47]) for point cloud operations, are written in C++. The third module, which associates source clouds with reference clouds, is a Python script. For compression, we use Draco [24] and LZMA [42]. To build the 3D map on board, we use Fast-LIO2 [68], a LiDAR SLAM algorithm. Then, we localize the point clouds in this 3D map using a normal distribution transform (NDT) [5]. We embed our compression and decompression modules for higher throughput in OpenMP library’s wrappers. Based on the number of cores available on a

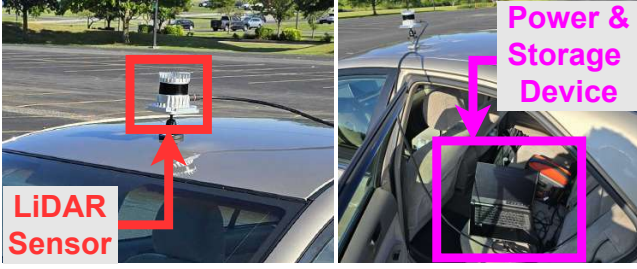


Figure 11: Real-world data collection setup.

machine, these can concurrently compress multiple source clouds. DejaView’s code and dataset are open-source<sup>2</sup>.

**Real-world Dataset.** To evaluate DejaView, we collected our own dataset by mounting an Ouster OS1-128 beam LiDAR [41] on top of an AV (Fig. 11). We drove daily on the same route around our campus for two months, accumulating over 297,000 point clouds. Using data for a single day from this dataset, we built a 3D map using FAST-LIO2 [68].

**Synthetic Traces.** We complemented our real-world dataset with synthetic LiDAR traces collected from CARLA [12], a photorealistic autonomous driving simulator. CARLA provides us with accurate ground truth for multiple applications that we do not have access to in the real world. For example, CARLA gives us accurate ground truth for localization, 3D object detection, and segmentation, which we use to evaluate DejaView. Moreover, CARLA enables us to perform a sensitivity analysis for DejaView by varying the number of channels and noise in a LiDAR and varying the traffic density. To match our real-world data collection, we collected 61 LiDAR traces along the same route, a total of more than 171,000 point clouds.

**Evaluation Platform.** We evaluated DejaView using a desktop with a 16-core Intel Xeon Silver 4114 CPU, 32 GB RAM, and Quadro P1000 GPU. This compute platform performs the compression and is connected to the internet through a wired network. We send the compressed point clouds to a CloudLab [14] server approximately 2000 miles away.

**Evaluation Metrics.** We use the following metrics to evaluate the compression and reconstruction error of DejaView.

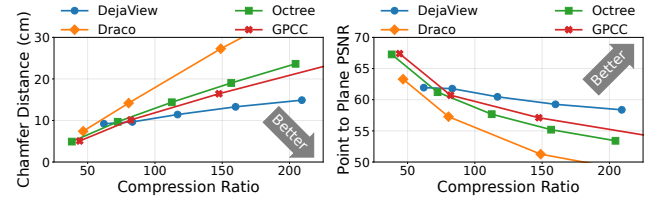
**Compression Ratio.** The compression ratio is the ratio of the original and compressed sizes source clouds. Higher compression ratios are better.

**Chamfer Distance.** We use the symmetric Chamfer distance [67] to measure DejaView’s reconstruction error. It is the least-mean-square-distance between every point in one point cloud to its nearest neighboring point in the other point cloud. More formally, given an original point cloud  $P$  and a reconstructed point cloud  $P'$ , the symmetric Chamfer distance is defined in Equation 7. A Chamfer distance of 0 indicates perfect reconstruction.

$$CD(P, P') = \frac{1}{|P|} \sum_i \min_j \|p_i - p'_j\|_2 \quad (6)$$

$$CD_{\text{sym}}(P, P') = (CD(P, P') + CD(P', P))/2 \quad (7)$$

**Point-to-Plane Peak Signal-to-Noise Ratio (PSNR).** The symmetric point-to-plane peak signal-to-noise ratio (PSNR) [57]



(a) DejaView ensures high compression ratios at lower Chamfer distances. (b) DejaView achieves higher compression ratios at high PSNRs.

Figure 12: Even at high compression ratios, DejaView is able to reconstruct the compressed point clouds accurately.

measures how well the points of one cloud align with the surfaces of the other cloud. Unlike the Chamfer distance, which measures the raw positions of 3D points, PSNR measures the retention of surface geometry information in the reconstructed point cloud.

## 4.2 End-to-end Experiments

We used an end-to-end implementation of DejaView for real-world experiments and evaluated its ability to compress point clouds against three baselines: the Point Cloud Library’s Octree-based compression [47], an open-source implementation of Draco [24] from Google, and geometry-based point cloud compression (GPCC) from MPEG [38]. We evaluated all four techniques at different levels of compression. All techniques have a knob that trades off compression for reconstruction error. For DejaView, we control the amount of compression using the *distance threshold*. Draco has a parameter to set the number of bits per point. GPCC controls compression with a quantization parameter. For Octree, we control the compression ratio using its octet resolution and point resolution.

In the first experiment, we measured the trade-off between compression ratio and reconstruction error (Chamfer distance and PSNR). Fig. 12a plots the impact of compression ratio on Chamfer distance for the four techniques using the real-world dataset. The ideal compression technique will have a high compression ratio for lower Chamfer distances (bottom right of Fig. 12a). Draco (orange line) can compress point clouds more than Octree (green line) and GPCC (red line), but does so by trading off Chamfer distance. Compared to Octree, Draco, and GPCC, DejaView achieves a significantly higher compression ratio at much lower Chamfer distances. For instance, for a 14 cm Chamfer distance, the compression ratios for GPCC, Octree, and Draco are 122, 112, and 80, respectively, whereas for DejaView, it is 220. Thus, *DejaView can compress source clouds 80% more than GPCC at the same Chamfer distance.*

As we increase the compression ratio for the four techniques, the Chamfer distance also increases. Ideally, we would like the increase in the Chamfer distance to be small otherwise, the reconstructed data would not be usable for downstream applications. By increasing GPCC’s compression ratio by 140, the Chamfer distance increases by 13 cm. This is undesirable. DejaView, on the other hand, trades off only 4 cm in Chamfer distance for increasing the compression ratio by 150. *Unlike the other three techniques, DejaView can ensure high compression by trading off small amounts of Chamfer distance (reconstruction error).*

For the same experimental setup, Fig. 12b plots point-to-plane PSNR as a function of the compression ratio. An ideal compression technique will have a high compression ratio for a high PSNR

<sup>2</sup>GitHub Repository: <https://github.com/nsslofficial/DejaView>

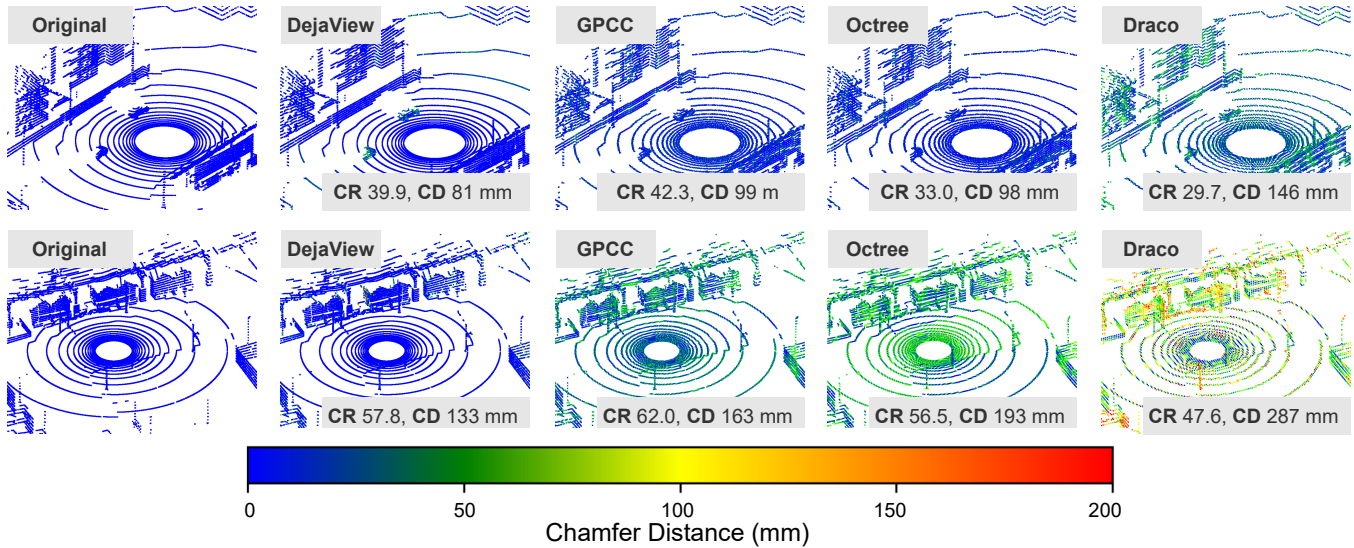


Figure 13: DejaView achieves lower reconstruction error, measured by CD, compared to GPCC, Octree, and Draco, particularly at higher CR.

i.e., it should operate on the top right of the graph. We observe a similar trend in this experiment, i.e., DejaView achieves a high compression ratio for low reconstruction error (or high PSNR). For a PSNR of 59, DejaView is doing approximately 98%, 120% and 175% better than GPCC, Octree and Draco, respectively. *This demonstrates that DejaView can preserve both the surface geometry information of source cloud and the positional information of the points at high compression levels, while GPCC, Draco, and Octree cannot.*

We also perform qualitative analysis by visualizing the compressed point clouds of the four techniques at different levels of compression in Fig. 13. We color-coded all points in the point clouds based on their distances from points in the ground-truth uncompressed point cloud on the left. A blue color indicating a lower Chamfer distance shows that the compression techniques preserve the 3D position of the given point with high accuracy. Other colors (green, yellow, and red) indicating a higher Chamfer distance show that the compression technique could not preserve the 3D position of the points. DejaView demonstrates superior capability to preserve the 3D positions of points, even at higher compression ratios, compared to GPCC, Draco, and Octree.

Recent deep learning-based methods, e.g., OctAttention [21], can achieve high compression ratios, but we exclude them from end-to-end experiments due to their significantly higher compression and decompression times. Instead, we analyze them separately in §4.6.

### 4.3 Application-level Results

AV-captured point clouds are crucial for training and testing perception modules such as localization, object detection, and semantic segmentation. In this section, we evaluate DejaView’s compression effects on these critical AV tasks.

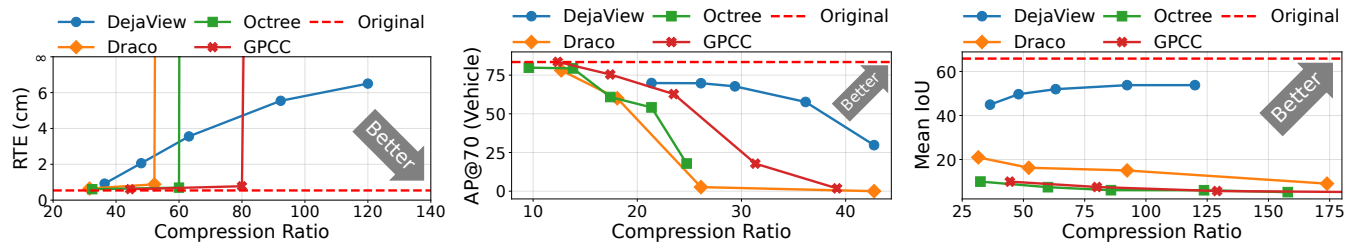
**Localization.** Localization determines the precise pose of an AV within a 3D map. AVs use NDT [5] to align the point cloud with the 3D map to estimate their pose. We evaluate the precision of localization using the relative translation error (RTE) [23], the root mean square distance between the ground truth and the estimated

positions. In this experiment, we use the synthetic dataset generated from CARLA, with which we have ground-truth 3D positions. Using data from a single day, we build a 3D map and then localize five point cloud sets on this map: raw uncompressed and those compressed by GPCC, Octree, Draco, and DejaView.

Fig. 14a shows the localization error (RTE) against the compression ratio for DejaView (blue), GPCC (orange), Octree (red), and Draco (green). The dotted red line shows the RTE for raw point clouds. An ideal compression scheme will have a high compression ratio and low localization error (RTE), i.e., bottom right of Fig. 14a. As the compression ratio increases, so do the Chamfer distance and localization errors. Higher Chamfer distances indicate a large deviation of compressed point positions from the raw data. Although NDT can accurately align raw point clouds, the deviation in point positions for compressed point clouds leads to higher localization errors. Of the four schemes, only DejaView’s localization error remains below 7 cm even with compression ratios up to 120. NDT cannot localize point clouds at all after a compression ratio of 50, 60, and 80 for Draco, Octree, and GPCC, respectively. *These results show that DejaView can achieve higher compression with minimal effect on end-to-end localization accuracy.*

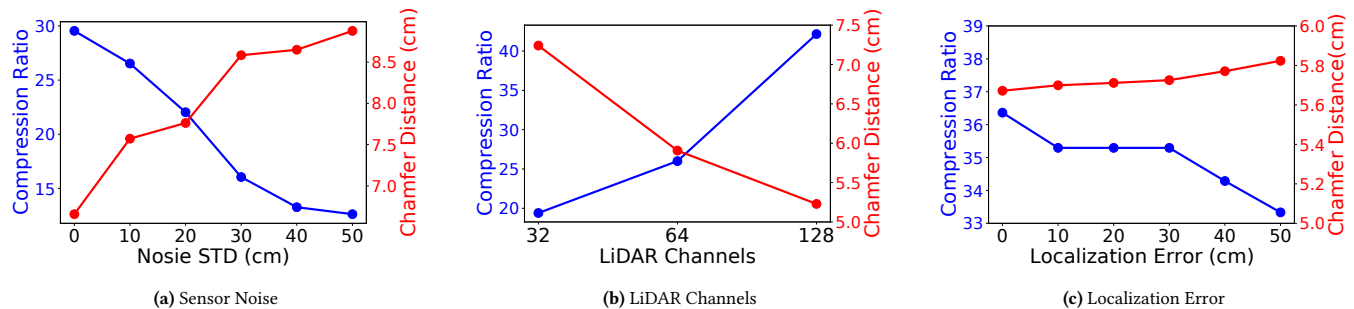
**Object Detection.** Object detection is a crucial component in the autonomous driving pipeline. We evaluate the effect of compression on the performance of a 3D object detection model, Part-A<sup>2</sup> net [52]. We trained this model for 100 epochs using 2,700 point clouds from CARLA. For testing, we used 3 days of data from CARLA, from which the first day’s data were used to build a 3D map and the second day’s data for a reference dataset. We compressed the third day data using DejaView, GPCC, Octree, and Draco.

Fig. 14b plots average precision (AP) at an intersection-over-union (IoU) threshold of 70 (AP@70) for AV bounding boxes as a function of compression ratio. AP@70 is the area under the precision-recall curve. A higher value of AP indicates better overall detection performance [16]. Ideally, a compression scheme should have high AP for high compression ratio i.e., top-right of Fig. 14b.



(a) DejaView achieves relatively accurate localization despite high compression. (b) DejaView ensures accurate object detection at high compression levels. (c) DejaView, despite high compression, achieves accurate segmentation.

**Figure 14:** Performance evaluation of DejaView on perception tasks against three baselines (Draco, Octree, and GPCC).



**Figure 15:** Sensitivity Analysis of DejaView.

As expected, AP drops with higher compression ratios. However, this drop is rapid for Octree, GPCC, and Draco. On the other hand, DejaView’s AP degrades gracefully with an increasing compression ratio. At AP values of 30, DejaView achieves 95%, 79%, and 48% better compression than Draco, Octree, and GPCC, respectively.

**3D Semantic Segmentation.** 3D semantic segmentation is another important task in the autonomous driving pipeline. In this task, each point in the point cloud is assigned a semantic label, e.g., road, car, tree, etc. (assigning object labels to individual points), we fine-tuned the last 9 layers of a MinkowskiNet-based model [7, 73] pre-trained on the Semantic KITTI dataset [3]. For this, we evaluated mean IoU (mIoU), across all class labels, a metric that determines the precision of semantic segmentation (formally defined in [3]). A technique that assigns correct labels to every point in the point cloud will have a mIoU of 100. Thus, a higher mIoU is better.

Fig. 14c shows mIoU at various compression ratios. An ideal compression technique will have high mIoU for high compression ratio, i.e., it will operate in the top right of Fig. 14c. Of these four schemes, DejaView’s mIoU scores are better for all compression ratios. Octree, GPCC, and Draco have low mIoU even at lower compression ratios, and it further decreases as the compression ratio increases. At a compression ratio of 122, DejaView achieves a 5x higher mIoU than Draco, while Octree and GPCC exhibit nearly zero mIoU. DejaView’s performance improves at higher compression ratios due to the reduced impact of Draco compression.

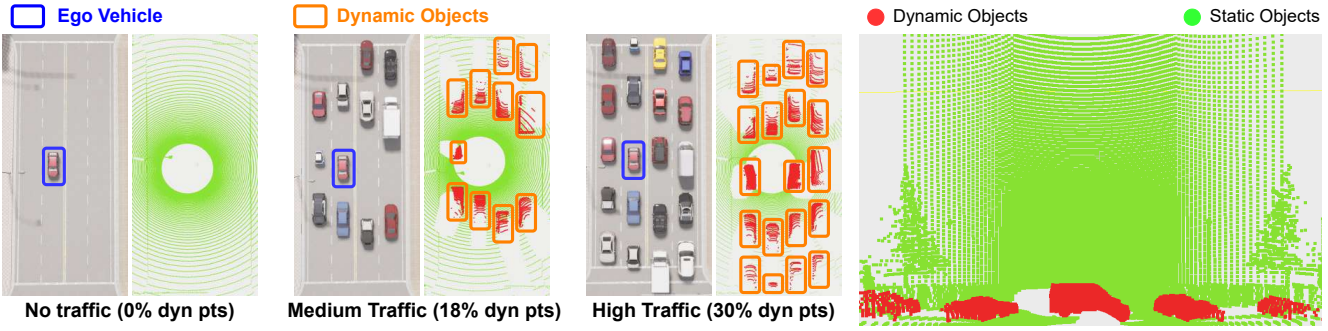
#### 4.4 Sensitivity Analysis

**Sensor Noise.** Real-world LiDAR data inherently contain sensor noise, which shifts 3D points from their actual positions. The magnitude of this noise varies, typically ranging from 1 cm to 10 cm

for commercial-off-the-shelf LiDAR sensors [26]. To evaluate the impact of sensor noise on DejaView, we simulate different noise levels in the point cloud using CARLA’s Gaussian noise model. We vary the standard deviation of LiDAR noise from 0 to 50 cm. In Fig. 15a, we plot the compression ratio (blue) and the Chamfer distance (red) as functions of the sensor noise for DejaView with the *distance threshold* set at 10 cm.

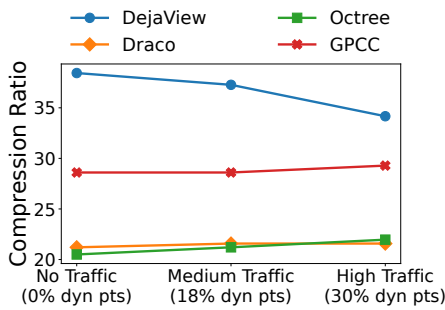
As the standard deviation of sensor noise increases from 0 cm to 50 cm, the compression ratio decreases from 30 to 13. At higher noise levels, the 3D points in source cloud deviate from their actual positions. Consequently, DejaView cannot associate them with points in reference cloud. As a result, these are instead stored as exclusive points. As noted before, increased exclusive points lead to a lower compression ratio. Moreover, this increases the Chamfer distance from 6.6 cm to 8.9 cm because more points are classified as exclusive and applying Draco compression to these points reduces their precision. To quantify Draco’s impact on the Chamfer distance, we compressed 2000 point clouds using DejaView with a 10 cm *distance threshold*. When decompressing the point clouds, we separate the points into those compressed by Draco and those stored as indices. Draco compressed points had a 3.3x higher Chamfer distance compared to those stored as indices. Hence, a higher number of exclusive points reduces both the compression ratio and the overall quality of the reconstruction.

**LiDAR Channels.** The number of channels (laser beams) in a LiDAR sensor impacts the point density (points per unit area) of the collected point clouds. To evaluate the effect of the number of LiDAR beams on DejaView, we generated CARLA data for 32, 64, and 128 beam LiDARs. In Fig. 15b, we plot the compression ratio (blue) and Chamfer distance (red) of DejaView as functions of the LiDAR channels. As the number of channels increases from 32 to 64,



(a) Bird-eye-view CARLA images and point clouds under different traffic conditions. The dynamic points peak at 30% when the ego-vehicle is fully surrounded by nearby AVs. (b) 2D rendering of a LiDAR’s vertical field-of-view shows dynamic objects only occupy a small portion of the point cloud.

**Figure 16:** Illustrations of traffic scenarios generated in CARLA for evaluating the effect of dynamic content percentage on compression ratio of DejaView.

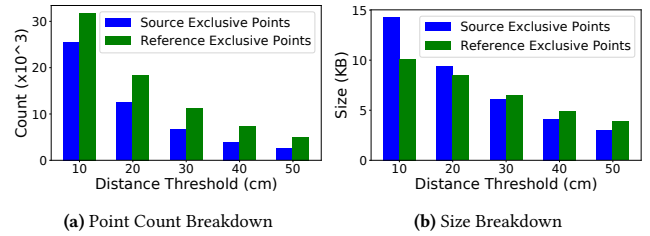


**Figure 17:** DejaView ensures high compression across all traffic conditions.

the compression ratio of DejaView also increases from 19 to 42. This is because, DejaView has a higher probability of finding common points (within the *distance threshold*) between source cloud and reference cloud with denser point clouds. This translates to fewer exclusive points and a higher compression ratio. Consequently, fewer exclusive points are subject to Draco’s quantization, reducing Chamfer distance by 2 cm.

**Localization Error.** Localization is a process in which AV uses sensor data to find its precise location in the world. This location information is then associated with the corresponding sensor data. As explained in §3, DejaView uses this location information to find a reference cloud for each source cloud. To evaluate the effect of localization error on DejaView’s performance, we generate data from CARLA along with the ground-truth location of each point cloud. To simulate the localization error, we add a random number sampled from the uniform distribution to the ground-truth location. We vary the range of the uniform distribution to control the magnitude of the error. Fig. 15c plots the compression ratio (blue) and the Chamfer distance (red) of DejaView versus the localization error. A localization error of 50 cm only reduces the compression ratio from 36 to 33 and increases the Chamfer distance by 0.1 cm. The slight reduction in compression ratio occurs because localization error can lead to suboptimal selection of a reference cloud for each source cloud. This, in turn, means fewer common points, thus affecting the compression ratio negatively.

**Dynamic Environment.** In this experiment, we systematically analyze the effect of the dynamic objects (vehicles, pedestrians, and cyclists, etc.) on DejaView’s compression ratio. As it is difficult to conduct controlled experiments in the real-world with dynamic



**Figure 18:** Comparison of point count and size breakdown of exclusive points from the source and reference clouds in DejaView’s compressed representation.

objects, for safety reasons and otherwise, we use CARLA for these evaluations. We generate three CARLA scenarios (Fig. 16a) with varying number of dynamic objects. The *no-traffic* scenario contains 0% dynamic points, *medium traffic* contains 18% dynamic points, and *high traffic* contains 30% dynamic points. Even in *high-traffic* scenarios, where the ego-vehicle is completely surrounded by dynamic objects, dynamic points only occupy 30% of the point cloud. This is because, although the AV is surrounded by objects along the horizontal field of view of LiDAR, these objects only occupy a small area on the vertical field of view (Fig. 16b).

Fig. 17 shows the effect of the number of dynamic objects on the compression ratio of DejaView and other baselines. DejaView consistently outperforms the three baselines. At lower traffic densities, DejaView achieves higher compression ratios because fewer dynamic objects result in fewer exclusive points in source cloud relative to reference cloud, which increases compression ratio for DejaView. In contrast, GPCC, Octree, and Draco show nearly constant compression ratios since they depend on the total number of points rather than the underlying scene dynamics.

**Distance Threshold.** Two points are considered identical in the source and reference clouds if they lie within a certain 3D distance of one another (*distance threshold*). We use *distance threshold* as a knob to control the tradeoff between compression ratio and reconstruction error. By increasing the *distance threshold* from 10 cm to 50 cm, DejaView trades off only 7 cm in Chamfer distance (or 5 units of PSNR) to improve the compression ratio from 30 to 66 (Fig. 7). Using this knob, system designers can tune DejaView for diverse application requirements, i.e., higher compression for applications that can tolerate low reconstruction quality, and vice versa.

**DejaView Compressed Representation.** The compressed representation produced by DejaView consists of exclusive points from source cloud and pointers to exclusive points in reference cloud. Fig. 18a and Fig. 18b provide a breakdown for the exclusive point count and corresponding compressed size as a function of the distance threshold. Both source cloud and reference cloud contain 76K points. At a distance threshold of 10 cm, DejaView identifies 25K exclusive points in the source cloud and 30K exclusive points in the reference cloud. The compressed representation of these points collectively requires only 23 KB of memory. Increasing the distance threshold to 50 cm further reduces the number of exclusive points, lowering the collective compressed size to only 7 KB.

To reconstruct the source cloud, DejaView stores pointers to exclusive points in the reference clouds. These pointers are the indices of exclusive points in the reference clouds. A single pointer is stored as an int32. However, a raw 3D point would require 12 bytes. Finally, DejaView applies LZMA compression to the sequence of indices for further compression.

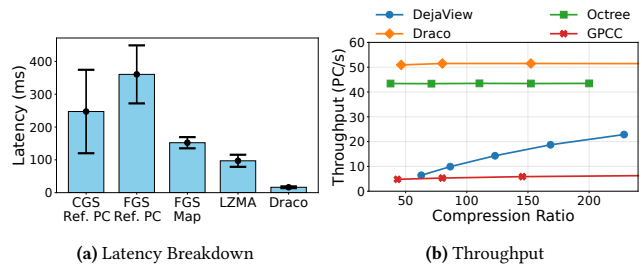
#### 4.5 Latency and Throughput Evaluations

Fig. 19a shows the latency breakdown per point cloud across different components of DejaView. The fine-grained search (FGS Ref. PC) and coarse-grained search (CGS Ref. PC) against the reference cloud are the most time-consuming stages, taking about 360 ms and 250 ms on average, respectively. Together, they account for roughly 69% of the total latency. Although the fine-grained search processes fewer points, it requires more time because it performs a detailed k-d tree lookup, whereas the coarse-grained search operates at a lower resolution. The fine-grained search with the map (FGS Map) is faster than the FGS Ref. PC because it matches a relatively smaller set of points. Draco and LZMA compression contribute only a small fraction of the total latency. DejaView offline builds the k-d tree structure from the 3D map. On average, for a 3D map of 1.5 GB, DejaView takes 17 seconds to build the k-d tree structure.

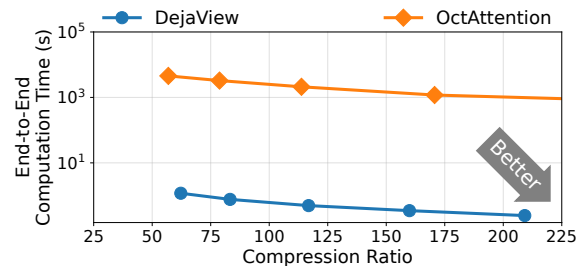
Fig. 19b shows the throughput of four techniques as a function of the compression ratio. Throughput is defined as the number of point clouds compressed and transferred to cloud storage per second. For GPCC, Draco, and Octree, the throughput remains relatively constant across compression ratios. In contrast, DejaView's throughput increases with higher compression ratios. This is because larger *distance thresholds* in DejaView identify more common points, reducing the number of exclusive points that require fine-grained searches. As a result, the compression latency decreases, leading to higher throughput. Despite this improvement, DejaView's throughput is still approximately 2.2× lower than Draco's, as it trades off latency to achieve better compression ratios.

#### 4.6 Comparison with Learning-based Methods

Recent learning-based point cloud compression techniques, such as OctAttention [21] and OctSqueeze [30], leverage octree representations for entropy encoding. These methods can achieve better reconstruction quality at a given compression ratio, but their time required to compress and decompress point clouds is very high [71]. To evaluate this tradeoff, we measured the compression ratio, reconstruction accuracy (Chamfer distance), and end-to-end computation



**Figure 19:** (a) Latency breakdown of different components of DejaView. (b) End-to-end throughput of DejaView increases with compression ratio, while it remains almost constant for all other compressors.



**Figure 20:** The y-axis is in logarithmic scale. A learning-based compression model (OctAttention) requires three orders of magnitude more time to compress and decompress point clouds relative to DejaView.

time, i.e., the sum of compression and decompression latency, for DejaView and OctAttention using over 2000 point clouds. We used the open source OctAttention model [20] pre-trained on the KITTI dataset [3]. To simulate a realistic deployment scenario, we attached a P1000 GPU to our edge computing platform for running inference. We plot the results for this experiment in Fig. 20. The x-axis represents the compression ratio. The y-axis, on a logarithmic scale, represents end-to-end computation time, which is the sum of compression and decompression times.

Across all compression ratios, on average, OctAttention achieves a higher reconstruction accuracy than DejaView, i.e., 5 cm lower Chamfer distance. For higher compression ratios, i.e., 200, the reconstruction accuracy difference is only 1 cm. However, this comes at the cost of end-to-end computation time. For a given compression ratio, the end-to-end computation time for OctAttention is approximately 3990 times more than DejaView (Fig. 20). Putting this number in perspective, for a compression ratio of 170, it takes DejaView 340 ms to compress and decompress a single point cloud, whereas, OctAttention takes almost 9 minutes to compress and decompress the same point cloud. This means that for one second of data from a single AV LiDAR, it would take OctAttention 1.5 hours to compress and decompress the 3D data. Although deep learning techniques achieve comparable compression ratios with marginally better reconstruction accuracy, their three orders of magnitude higher latency renders them impractical for the time-sensitive use cases. In contrast, DejaView trades a small amount of reconstruction accuracy for faster computation times and comparable compression ratios.

## 5 Related Work

**Traditional Methods.** Traditional compression algorithms leverage spatial redundancies in point clouds using tree structures.

Among these, Octree is the widely used [6, 48, 49]. Octree-based methods quantize the point cloud and then recursively subdivide it into eight cubes and encode their occupancy for compression. In contrast, DejaView employs an Octree structure for coarse-grained *diff* operation but mitigates the artifacts caused by quantization and binning through a fine-grained *diff* operation that refines the results at a point level. Furthermore, Octree representations have also been utilized to capture spatial redundancies between consecutive point clouds to improve compression ratios [10, 13, 22, 33, 62]. In contrast, DejaView identifies spatial redundancies across larger temporal scales by finding a reference cloud in the vicinity of source cloud, to achieve a higher compression ratio.

While Octree-based methods are effective for sparse point clouds, k-d tree-based approaches are often preferred for denser point clouds. Unlike the octree, a k-d tree is primarily used to reorder points so that spatially adjacent points are stored close together in memory. This spatial reordering improves the compression ratio by enhancing the performance of entropy and predictive coding techniques [11]. The open-source compression library Draco from Google [24] leverages the k-d tree for this purpose. In contrast, DejaView uses k-d tree for efficiently computing fine-grained *diff* between point clouds.

**Learning-based Methods.** Early learning-based methods voxelize point clouds and use convolutional autoencoders to encode the 3D grids [45, 60, 61]. However, voxelization is computationally expensive and memory-intensive, as it allocates voxels even for empty regions. To address this limitation, subsequent works, such as PointNet and PointNet++, directly process raw point clouds without voxelization [31, 43, 44, 70]. Another line of research projects point clouds into images, which are then compressed using image-based techniques [1, 18, 25, 28, 58, 59, 74]. While these methods effectively exploit local spatial and temporal correlations, they primarily capture short-term redundancies between consecutive frames. In contrast, DejaView identifies and leverages long-term spatial and temporal redundancies, achieving higher compression ratios without the high computational cost of voxelization or network training.

Hybrid methods combine deep learning with tree structures, such as octrees. These approaches enhance compression by learning entropy models that exploit the spatial and hierarchical context of octree nodes, including their level, occupancy, and neighborhood relationships [9, 21, 30, 46, 54]. While such methods achieve high compression ratios, they incur significant computational overhead and require training separate models for each dataset, limiting their real-time applicability and generalization [71]. In contrast, DejaView leverages hierarchical representations to efficiently compute differences between point clouds without the overhead of model training, while maintaining generalizability.

**LiDAR-based SLAM.** SLAM processes a stream of point clouds to estimate a AV's trajectory and maintains a single consistent 3D map of the environment [68, 69]. To achieve this, SLAM continuously performs point cloud-to-map alignment to refine AV pose. In doing so, it also computes residual differences between the current point cloud and the existing map. Of these differences, it integrates static and stable 3D points into the map. SLAM discards unstable and dynamic points (such as those belonging to vehicles, pedestrians, etc.).

Consequently, it does not preserve raw point clouds and cannot reconstruct them from the processed 3D map.

Unlike SLAM, DejaView does not distinguish between static and dynamic points. Instead, it performs a bidirectional *diff* operation to identify common and exclusive points. It stores references to common points and compactly encodes exclusive points. DejaView preserves every point cloud, including dynamic content, in a fully reconstructible form. This enables downstream applications, such as forensic replay, safety validation, and retraining of perception systems, and can even provide data to build SLAM maps.

## 6 Discussion and Future Work

Existing approaches exploit redundancies across short temporal intervals (i.e., between consecutive frames) to compress point clouds. In contrast, DejaView leverages a 3D map and spatially proximate point clouds from a reference dataset to identify redundancies over much larger temporal scales (days, months, or even years), achieving significantly higher compression ratios. DejaView performs best when the source data closely matches the reference dataset and 3D map. However, changes such as road maintenance, construction, or building demolition can render the reference dataset and 3D map stale, thereby reducing the compression ratio. Regularly updating the reference dataset and 3D map can help maintain high compression performance. For previously compressed data, DejaView can adopt one of the two strategies: it can either recompress the data with respect to the updated reference dataset and 3D map, or maintain separate reference datasets and maps for different sets of source data. We leave an in-depth examination of these and other innovative approaches to future work.

The selection of point clouds that make up the reference dataset provides another avenue for improvement. In this paper, we used the first 10 days of collected data to build the reference dataset. Future work can explore intelligent point cloud selection techniques, across space and time, that optimize compression ratio and reduce, if not eliminate, the need to update the reference dataset.

## 7 Conclusion

In this paper, we introduce DejaView, a framework for compressing point cloud data collected over large timescales, i.e., weeks, months, or years. DejaView key insight is to use spatio-temporal correlations, i.e., the fact that environmental geometry remains relatively static across time. It introduces an efficient *diff* operation which isolates temporal redundancies to build a compact representation. We evaluated DejaView against state-of-the-art methods using two datasets: a real-world set collected with an Ouster OS1-128 LiDAR and a simulated set from the CARLA autonomous driving simulator. DejaView achieves 2.5× higher compression ratio compared to existing techniques and better preserves the performance of downstream tasks, such as localization, 3D object detection, and 3D semantic segmentation.

## Acknowledgments

We thank the anonymous reviewers and shepherd for their insightful comments. This work is supported in part by the National Science Foundation (NSF) under award numbers CNS-2348461, 2411386/2411387, 2106634/2106635.

## References

- [1] Khartik Ainala, Rufael N Mekuria, Birendra Khathariya, Zhu Li, Ye-Kui Wang, and Rajan Joshi. 2016. An improved enhancement layer for octree based point cloud compression with plane projection approximation. In *Applications of Digital Image Processing XXXIX*, Vol. 9971. SPIE, 223–231.
- [2] Amazon Web Services. 2024. AWS Pricing Calculator - Amazon S3. <https://calculator.aws/#/createCalculator/S3> Accessed: 2024-12-06.
- [3] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jurgen Gall. 2019. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *Proceedings of the IEEE/CVF international conference on computer vision*. 9297–9307.
- [4] Julie Stephany Berrio, Stewart Worrall, Mao Shan, and Eduardo Nebot. 2022. Long-Term Map Maintenance Pipeline for Autonomous Vehicles. *IEEE Transactions on Intelligent Transportation Systems* 23, 8 (2022), 10427–10440. <https://doi.org/10.1109/ITITS.2021.3094485>
- [5] Peter Biber and Wolfgang Straßer. 2003. The normal distributions transform: A new approach to laser scan matching. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)* (Cat. No. 03CH37453), Vol. 3. IEEE, 2743–2748.
- [6] Anthony Chen, Shiwen Mao, Zhu Li, Minrui Xu, Hongliang Zhang, Dusit Niyato, and Zhu Han. 2023. An introduction to point cloud compression standards. *GetMobile: Mobile Computing and Communications* 27, 1 (2023), 11–17.
- [7] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 2019. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 3075–3084.
- [8] Ry Crist and Steve Conaway. 2024. How Fast Is Wi-Fi 6? <https://www.cnet.com/home/internet/how-fast-is-wi-fi-6/> Accessed: Dec. 8, 2024.
- [9] Mingyue Cui, Junhua Long, Mingjian Feng, Boyang Li, and Huang Kai. 2023. OctFormer: Efficient octree-based transformer for point cloud compression with local enhancement. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 470–478.
- [10] Ricardo L De Queiroz and Philip A Chou. 2016. Compression of 3D point clouds using a region-adaptive hierarchical transform. *IEEE Transactions on Image Processing* 25, 8 (2016), 3947–3956.
- [11] Olivier Devillers and P-M Gandoin. 2000. Geometric compression for interactive transmission. In *Proceedings Visualization 2000. VIS 2000* (Cat. No. 00CH37145). IEEE, 319–326.
- [12] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*. 1–16.
- [13] Antoine Dricot and João Ascenso. 2019. Hybrid octree-plane point cloud geometry coding. In *2019 27th European Signal Processing Conference (EUSIPCO)*. IEEE, 1–5.
- [14] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. 1–14. <https://www.flux.utah.edu/paper/duplyakin-atc19>
- [15] Klaus Engel, Markus Hadwiger, Joe M Kniss, Aaron E Lefohn, Christof Rezk Salama, and Daniel Weiskopf. 2004. Real-time volume graphics. In *ACM Siggraph 2004 Course Notes*. 29–es.
- [16] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. 2010. The pascal visual object classes (voc) challenge. *International journal of computer vision* 88 (2010), 303–338.
- [17] Xiaohua Feng, Edward Swarlat Dawam, and Dayou Li. 2019. Autonomous vehicles' forensics in smart cities. In *2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UC/ATC/CBDCOM/IOP/SCI)*. IEEE, 1688–1694.
- [18] Yu Feng, Shaoshan Liu, and Yuhao Zhu. 2020. Real-time spatio-temporal lidar point cloud compression. In *2020 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 10766–10773.
- [19] Jannik Fritsch, Tobias Kuehnl, and Andreas Geiger. 2013. A New Performance Measure and Evaluation Benchmark for Road Detection Algorithms. In *International Conference on Intelligent Transportation Systems (ITSC)*.
- [20] Chunyang Fu. 2023. OctAttention: Octree-Based Large-Scale Contexts Model for Point Cloud Compression. <https://github.com/zb12138/OctAttention>. Accessed: 2025-06-30.
- [21] Chunyang Fu, Ge Li, Rui Song, Wei Gao, and Shan Liu. 2022. Octattention: Octree-based large-scale contexts model for point cloud compression. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 36. 625–633.
- [22] Diogo C Garcia and Ricardo L de Queiroz. 2018. Intra-frame context-based octree coding for point-cloud geometry. In *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, 1807–1811.
- [23] Andreas Geiger, Philip Lenz, and Raquel Urtasun. 2012. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 3354–3361.
- [24] google. 2017. DRACO: 3D data compression. <https://github.com/google/draco>.
- [25] Yingshen He, Ge Li, Yiting Shao, Jing Wang, Yueru Chen, and Shan Liu. 2020. A point cloud compression framework via spherical projection. In *2020 IEEE International Conference on Visual Communications and Image Processing (VCIP)*. IEEE, 62–65.
- [26] Hanno Holzhüter, Jörn Bödewadt, Shima Bayesteh, Andreas Aschinger, and Holger Blume. 2023. Technical concepts of automotive LiDAR sensors: a review. *Optical Engineering* 62, 3 (2023), 031213–031213.
- [27] Mohammad Aminul Hoque and Ragib Hasan. 2021. Avguard: A forensic investigation framework for autonomous vehicles. In *ICC 2021-IEEE International Conference on Communications*. IEEE, 1–6.
- [28] Hamidreza Houshiar and Andreas Nüchter. 2015. 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, 1–8.
- [29] Zhihao Hu, Zhenghao Chen, Dong Xu, Guo Lu, Wanli Ouyang, and Shuhang Gu. 2020. Improving deep video compression by resolution-adaptive flow coding. In *Computer Vision—ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*. Springer, 193–209.
- [30] Lila Huang, Shenlong Wang, Kelvin Wong, Jerry Liu, and Raquel Urtasun. 2020. Octsqueeze: Octree-structured entropy model for lidar compression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 1313–1323.
- [31] Tianxin Huang and Yong Liu. 2019. 3d point cloud geometry compression on deep learning. In *Proceedings of the 27th ACM international conference on multimedia*. 890–898.
- [32] Satish Jeyachandran. 2024. Meet the 6th-generation Waymo Driver: Optimized for costs, designed to handle more weather, and coming to riders faster than before. <https://waymo.com/blog/2024/08/meet-the-6th-generation-waymo-driver/>.
- [33] Julius Kammerl, Nico Blodow, Radu Bogdan Rusu, Suat Gedikli, Michael Beetz, and Eckehard Steinbach. 2012. Real-time compression of point cloud streams. In *2012 IEEE international conference on robotics and automation*. IEEE, 778–785.
- [34] Insup Kim, Ganggyu Lee, Seyoung Lee, and Wonsuk Choi. 2022. Data Storage System Requirement for Autonomous Vehicle. In *2022 22nd International Conference on Control, Automation and Systems (ICCAS)*. 45–49. <https://doi.org/10.23919/ICCAS55662.2022.10003785>
- [35] Meiqin Liu, Chenming Xu, Yukai Gu, Chao Yao, Weisi Lin, and Yao Zhao. 2024. I<sup>2</sup>VC: A Unified Framework for Intra- & Inter-frame Video Compression. *arXiv preprint arXiv:2405.14336* (2024).
- [36] Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Chunlei Cai, and Zhiyong Gao. 2019. Dvc: An end-to-end deep video compression framework. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 11006–11015.
- [37] Motional. 2020. nuScenes. <https://www.nuscenes.org/nuscenes>.
- [38] MPEG. 2025. MPEG Geometry-based Point Cloud Compression (G-PCC) Reference Software. <https://github.com/MPEGGroup/mpeg-pcc-tmc13>
- [39] Khan Muhammad, Amin Ullah, Jaime Lloret, Javier Del Ser, and Victor Hugo C de Albuquerque. 2020. Deep learning for safe autonomous driving: Current challenges and future directions. *IEEE Transactions on Intelligent Transportation Systems* 22, 7 (2020), 4316–4336.
- [40] Ali Nasser, Adriaan Schiphorst, Norman di Palo, Jordan Sotudeh, Fazel Chaudry, Jeremy Horne, Drue Freeman, Mark A. Crawford Jr., Akbar Ladak, Shlomit Hacohen, Zeljko Medenica, Maxime Flament, Joakim Svennson, William Morris, Matthew Nancekievill, Bureau Merkwwaardig, Sabina Begovic, and Benedict Redgrove. 2020. 2020 Autonomous Vehicle Technology Report. <https://www.wevolver.com/article/2020.autonomous.vehicle.technology.report>.
- [41] Ouster. 02023. OS1 LiDAR Sensor. <https://ouster.com/products/hardware/os1-lidar-sensor> Accessed: 2024-09-03.
- [42] PM Parekar and SS Thakare. 2014. Lossless data compression algorithm—a review. *International Journal of Computer Science & Information Technologies* 5, 1 (2014), 276–278.
- [43] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 652–660.
- [44] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. 2017. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems* 30 (2017).
- [45] Maurice Quach, Giuseppe Valenzise, and Frederic Dufaux. 2019. Learning convolutional transforms for lossy point cloud geometry compression. In *2019 IEEE international conference on image processing (ICIP)*. IEEE, 4320–4324.
- [46] Zizheng Que, Guo Lu, and Dong Xu. 2021. Voxelcontext-net: An octree based framework for point cloud compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6042–6051.
- [47] Radu Bogdan Rusu and Steve Cousins. 2011. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, Shanghai, China.
- [48] Ruwen Schnabel and Reinhard Klein. 2006. Octree-based Point-Cloud Compression. In *Eurographics Symposium on Point-Based Graphics*. 111–120. <https://doi.org/10.2312/SPBG/SPBG06/111-120>

- [49] Sebastian Schwarz, Marius Preda, Vittorio Baroncini, Madhukar Budagavi, Pablo Cesar, Philip A Chou, Robert A Cohen, Maja Krivokuća, Sébastien Lasserre, Zhu Li, et al. 2018. Emerging MPEG standards for point cloud compression. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 1 (2018), 133–148.
- [50] Prinkle Sharma and James Gillanders. 2022. Cybersecurity and forensics in connected autonomous vehicles: A review of the state-of-the-art. *IEEE Access* 10 (2022), 108979–108996.
- [51] Prinkle Sharma, Umesh Siddanagaiah, and Gökhan Kul. 2020. Towards an AI-based after-collision forensic analysis protocol for autonomous vehicles. In *2020 IEEE Security and Privacy Workshops (SPW)*. IEEE, 240–243.
- [52] Shaoshuai Shi, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. 2020. 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* 43, 8 (2020), 2647–2664.
- [53] Christina Suyong Shin, Weiwu Pang, Chuan Li, Fan Bai, Fawad Ahmad, Jeongyeup Paek, and Ramesh Govindan. 2024. RECAP: 3D traffic reconstruction. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*. 1252–1267.
- [54] Rui Song, Chunyang Fu, Shan Liu, and Ge Li. 2023. Efficient hierarchical entropy model for learned point cloud compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14368–14377.
- [55] Torsten Suel. 2019. Delta compression techniques. *Encyclopedia of Big Data Technologies* 63 (2019).
- [56] The Waymo Team. 2020. The Waymo Driver Handbook: How our highly-detailed maps help unlock new locations for autonomous driving. <https://waymo.com/blog/2020/09/the-waymo-driver-handbook-mapping/>.
- [57] Dong Tian, Hideaki Ochimizu, Chen Feng, Robert Cohen, and Anthony Vetro. 2017. Geometric distortion metrics for point cloud compression. In *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE, 3460–3464.
- [58] Chenxi Tu, Eijiro Takeuchi, Alexander Carballo, and Kazuya Takeda. 2019. Point cloud compression for 3d lidar sensor using recurrent neural network with residual blocks. In *2019 international conference on robotics and automation (ICRA)*. IEEE, 3274–3280.
- [59] Chenxi Tu, Eijiro Takeuchi, Chiyomi Miyajima, and Kazuya Takeda. 2016. Compressing continuous point cloud data using image compression methods. In *2016 IEEE 19th international conference on intelligent transportation systems (ITSC)*. IEEE, 1712–1719.
- [60] Jianqiang Wang, Dandan Ding, Zhu Li, and Zhan Ma. 2021. Multiscale point cloud geometry compression. In *2021 Data Compression Conference (DCC)*. IEEE, 73–82.
- [61] Jianqiang Wang, Hao Zhu, Haojie Liu, and Zhan Ma. 2021. Lossy point cloud geometry compression via end-to-end learning. *IEEE Transactions on Circuits and Systems for Video Technology* 31, 12 (2021), 4909–4923.
- [62] Miaohui Wang, Runnan Huang, Wuyuan Xie, Zhan Ma, and Siwei Ma. 2025. Compression Approaches for LiDAR Point Clouds and Beyond: A Survey. *ACM Transactions on Multimedia Computing, Communications and Applications* (2025).
- [63] Yuxin Wang, Yuankai He, Ruijun Wang, and Weisong Shi. 2024. Quantitative Analysis of Storage Requirement for Autonomous Vehicles. In *Proceedings of the 16th ACM Workshop on Hot Topics in Storage and File Systems* (Santa Clara, CA, USA) (*HotStorage '24*). Association for Computing Machinery, New York, NY, USA, 71–78. <https://doi.org/10.1145/3655038.3665948>
- [64] Waymo. 2021. Simulation City: Introducing Waymo's most advanced simulation system yet for autonomous driving. <https://waymo.com/blog/2021/07/simulation-city>. Accessed: 2025-10-25.
- [65] Waymo. 2024. About – Waymo Open Dataset. <https://waymo.com/open/about/>.
- [66] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. 2003. Overview of the H. 264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology* 13, 7 (2003), 560–576.
- [67] Tong Wu, Liang Pan, Junzhe Zhang, Tai Wang, Ziwei Liu, and Dahua Lin. 2021. Balanced chamfer distance as a comprehensive metric for point cloud completion. *Advances in Neural Information Processing Systems* 34 (2021), 29088–29100.
- [68] Wei Xu, Yixi Cai, Dongjiao He, Jiarong Lin, and Fu Zhang. 2022. Fast-lio2: Fast direct lidar-inertial odometry. *IEEE Transactions on Robotics* 38, 4 (2022), 2053–2073.
- [69] Wei Xu and Fu Zhang. 2021. Fast-lio: A fast, robust lidar-inertial odometry package by tightly-coupled iterated kalman filter. *IEEE Robotics and Automation Letters* 6, 2 (2021), 3317–3324.
- [70] Wei Yan, Shan Liu, Thomas H Li, Zhu Li, Ge Li, et al. 2019. Deep autoencoder-based lossy geometry compression for point clouds. *arXiv preprint arXiv:1905.03691* (2019).
- [71] Kang You, Tong Chen, Dandan Ding, M Salman Asif, and Zhan Ma. 2025. Reno: Real-time neural compression for 3d lidar point clouds. In *Proceedings of the Computer Vision and Pattern Recognition Conference*. 22172–22181.
- [72] Yang You, Yujing Lou, Qi Liu, Yu-Wing Tai, Lizhuang Ma, Cewu Lu, and Weiming Wang. 2020. Pointwise rotation-invariant network with adaptive sampling and 3d spherical voxel convolution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 12717–12724.
- [73] Liu Youquan, Bai Yeqi, Kong Lingdong, Chen Runnan, Hou Yuenan, Shi Botian, and Li Yikang. 2023. OpenPCSeg: An Open Source Point Cloud Segmentation Codebase. <https://github.com/PJLab-ADG/PCSeg>.
- [74] Xuanyu Zhou, Charles R Qi, Yin Zhou, and Dragomir Anguelov. 2022. Riddle: Lidar data compression with range image deep delta encoding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 17212–17221.