DUSTER: A Deduplication Framework for Efficient Point Cloud Storage and Retrieval [Technical Report]

Anonymous Author(s)

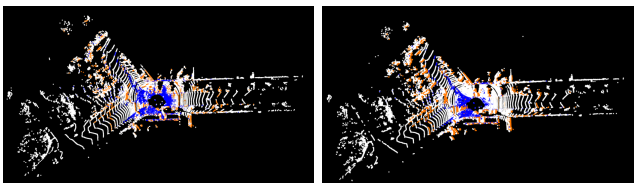
Abstract

The rise of embodied AI, exemplified by autonomous driving, has led to a rapid surge in the volume of point cloud (PC) data, highlighting the need for a dedicated storage and management system. In such scenarios, LiDAR sensors continuously generate PC frames at a stable frequency. As consecutive frames often capture overlapping regions, they exhibit spatio-temporal redundancy, resulting in duplicated 3D points in both the relative coordinate system (centered on the sensor) and the absolute coordinate system (aligned with global positioning). To address this issue, we propose DUSTER, a deduplication framework for efficient point cloud storage and retrieval. DUSTER considers the entire PC sequence for joint deduplication and employs a segmentation module to determine which frames should be grouped and processed together. For each identified segment, a deduplication module effectively detects and eliminates redundant points across the grouped PC frames. Finally, an indexing module manages the deduplicated data and reconstructs the original PC frames on demand during retrieval. Experimental results show that DUSTER achieves up to 42% reduction in the number of stored PC points and up to 80% reduction in disk space usage. Moreover, it supports efficient random and sequential PC frame retrieval in terms of I/O cost, while maintaining deep model prediction accuracy on reconstructed PC frames with minimal degradation.

1 Introduction

The rapid development of embodied AI [9] including applications such as autonomous driving [38], service robots [30], and drones has led to a deluge of point cloud (PC) data. Equipped with the LiDAR sensor, a single agent can generate up to 20 PC frames per second, each containing over 10^5 three-dimensional spatial points along with associated features such as light intensity [12]. This amounts to more than 130 GB of data per hour per agent. At fleet scale, for instance with over 10,000 vehicles deployed, the daily volume can reach the petabyte level [17], making efficient point cloud data storage and retrieval extremely challenging. With the continued growth of embodied AI applications, such as service robots and drones that also rely on LiDAR for perception [45], these challenges are expected to intensify further.

In aforementioned scenarios, the LiDAR sensor continuously produces a sequence of PC frames, each associated with a generation timestamp. It is observed that consecutive PC frames often exhibit spatio-temporal redundancy that they capture overlapping regions of the surrounding environment [33], thereby generating redundant points. An example of spatio-temporal redundancy between two consecutive PC frames is illustrated in Fig.1, where over 50% of the points in each PC frame are identified as redundant. This redundancy presents a critical opportunity for optimization: eliminating duplicated information can yield substantial storage savings without sacrificing data fidelity. Recent studies, such as MAST [26], have demonstrated the potential of managing and analyzing PC sequences on the server side. Efficient management of PC frame sequences requires optimizing storage consumption to allow more data to be stored and analyzed within available disk space. Detecting and removing redundancy across consecutive PC frames can significantly reduce the number of points that need to be stored on the server, thereby improving storage efficiency without relying on lossy frame-level downsampling (e.g., storing only one PC frame per second) [31]. Despite this opportunity, existing systems and processing libraries, including pgPointCloud [32] (an extension of PostgreSQL [34]), Ganos [7, 48], and PCL [33], do not support spatio-temporal redundancy removal in their storage or processing pipelines, resulting in sub-optimal storage efficiency. This highlights the need for a dedicated deduplication framework tailored to sequential PC data.



(a) Consecutive PC frame 1. (b) Consecutive PC frame 2.

Figure 1: An example of redundancy between two consecutive PC frames. Points in blue (resp. orange) represent redundant points in the relative (resp. absolute) coordinate system.

To develop such a framework with effective and efficient PC redundancy deduplication, three key challenges need to be solved:

First, spatio-temporal redundancy arises not only between two consecutive PC frames, but also across a broader window of neighboring frames captured within a short time interval. This observation motivates the need for jointly considering deduplication on arbitrary number of consecutive PC frames, rather than limiting to frame pairs [23], to maximize storage savings. However, finding an optimal deduplication strategy is non-trivial. Specifically, we study the multi-frame joint deduplication problem, which aims to maximize the number of eliminated redundant points when processing N consecutive frames, and prove it is is NP-hard [24] for $N \geq 3$. Therefore, a practical and efficient solution is essential.

Second, the massive daily volume of PC data, frequently amounting to petabytes particularly in scenarios involving the deployment of tens of thousands of vehicles or robots within an urban environment, renders large-scale deduplication prohibitively expensive. However, resources assigned for storage optimization and data preprocessing are often limited in practice [2, 14]. For example, commercial cloud-native systems such as Ganos report that preprocessing and optimization can incur substantial overheads [48]. Thus, given a sequence of PC frames, the key challenge is to design deduplication framework that deliver high-quality redundancy elimination while keeping the computation within limited budget.

Third, an effective data deduplication framework must achieve a dual objective: significant storage reduction and efficient data retrieval. We identify that deduplication across PC frames, while improving compression, introduces substantial data reconstruction overhead during retrieval. As our experiments will demonstrate, this overhead can lead to prohibitive I/O costs. This underscores the critical need for a retrieval-aware deduplication design that minimizes reconstruction overhead while preserving storage efficiency and query performance for downstream applications.

In this paper, we present DUSTER, a multi-frame, multi-coordinate system deduplication framework for efficient point cloud storage and retrieval. The framework performs joint deduplication across PC frame sequences, eliminates redundancy in both relative and absolute coordinate systems under a time budget, and optimizes storage while enabling efficient data retrieval with low I/O cost.

To enable redundancy removal across arbitrary numbers of consecutive PC frames, we introduce a deduplication module based on a point-wise matching algorithm that efficiently identifies redundant points across multiple frames. Specifically, we design an approach that recursively detects redundant clusters, starting from larger ones (e.g., 3D points shared by all frames) and then progressing to smaller ones. Redundant points efficiently identified through a series of redundancy detections across pairs of PC frames. The matched clusters are aggregated into a compact subset of points. By storing the spatial coordinate of each cluster only once, DUSTER effectively eliminates redundancy and optimizes storage.

As the number of PC frames grows, the computational cost of joint deduplication increases rapidly, while the storage gain saturates beyond a small number of frames. Given limited computational resources, deduplicating all frames in a sequence at once becomes inefficient. To address this, we partition the PC sequence into consecutive segments, each processed independently to achieve fast deduplication with comparable storage reduction. Specifically, we first apply integer programming to determine a segmentation plan that specifies the number of segments of each length (e.g., a plan may include 100 two-frame segments and 50 three-frame segments). We then design a learning-based multi-armed bandit algorithm [22] that assigns longer segments to regions likely to yield higher deduplication performance. The algorithm offers potentially higher deduplication gains by intelligently updating the sampling distribution based on reward feedback. This approach enables adaptive, cost-aware segmentation that achieves high deduplication efficiency while adhering to the available resource constraint.

Finally, DUSTER achieves efficient PC retrieval by managing and storing the deduplicated data at the granularity of point subsets. During data retrieval, when a target PC frame is requested, DUSTER reconstructs the original PC frame by loading and combining only the relevant point subsets, minimizing unnecessary disk access. We maintain an index of metadata to ensure that point subsets are correctly transformed and reassembled. A desirable feature of DUSTER is that it is designed to be orthogonal to existing compression techniques, and thus they can be applied to further provide additional storage savings. We demonstrate this by by integrating LASzip compression [20] into DUSTER.

We summarize our contribution as follows:

- We develop DUSTER, a deduplication framework for efficient point cloud storage and retrieval. To the best of our knowledge, DUSTER is the first framework that performs spatio-temporal redundancy deduplication on PC sequences to optimize storage consumption.
- We prove that the optimal PC deduplication problem is NP-hard. To tackle this, we develop a segmentation module that partitions the PC sequence into consecutive segments, a deduplication module that detects and removes redundancy within each segment, and an indexing module that manages the deduplicated data and enables efficient frame retrieval.
- Extensive experiments are conducted on real PC datasets. The results demonstrate that DUSTER achieves up to 42% reduction in the number of PC points stored and 80% reduction in required disk space. Moreover, DUSTER supports efficient random and sequential PC frame retrieval with reduced I/O cost, while maintaining deep model prediction accuracy on the reconstructed PC frames with minimal degradation.

The rest of this paper is organized as follows. In Section 2, we introduce the preliminaries and define the design objective. We introduce the related work in Section 3. Section 4 introduces the architecture of DUSTER. Section 5 introduces the detailed methodology. Finally, we evaluate the experimental results in Section 6.

2 Preliminaries

We proceed to introduce the preliminaries and the main design objective of this paper.

We first define the PC frame and the PC sequence. A point cloud frame P is defined as a set of multidimensional points $\mathsf{P} = \{p_1, \ldots, p_n\}$. Each $p = (x, y, z, F)$, where (x, y, z) denotes the three dimensional spatial coordinate, and F represents additional features of the point. In this work, we consider F to be the light intensity.

We then define a PC sequence $\mathcal{P}$ consisting of multiple consecutive PC frames, each associated with metadata including its timestamp and pose: $\mathcal{P} = \{(\mathsf{P}_1, t_1, pose_1), \ldots, (\mathsf{P}_m, t_m, pose_m)\}$, where the timestamp t_i indicates the generating time of P_i, and $pose_i$ denotes the pose of the LiDAR sensor when generating P_i. The pose is represented as a transformation matrix that maps the PC frame from the relative coordinate system (centered on the LiDAR sensor) to the absolute coordinate system (aligned with global positioning, where all PC frames are registered).

Data Retrieval Strategies. Two PC data retrieval strategies are considered, namely *random PC frame retrieval* and *sequential PC frame retrieval*. In this paper, we evaluate the performance of both retrieval strategies in terms of I/O cost.

Random PC Frame Retrieval. We define the random PC frame retrieval when a single PC frame is selected and loaded to the memory from disk. This retrieval happens when the user selects a PC frame for analytical tasks such as object prediction [26].

Sequential PC Frame Retrieval. We define the sequential PC frame retrieval when multiple *consecutive* PC frames are required to be loaded to memory. This typically occurs when multiple consecutive PC frames are loaded for tasks such as visualization.

Design Objective. We define the objective of designing the deduplication framework for PC data as follows:

Design Objective 1. *We aim to design a framework for PC sequences that maximizes spatio-temporal redundancy elimination to reduce storage costs under limited computational resources, while*

enabling efficient data retrieval (in terms of I/O cost) and preserving high-quality PC frame reconstruction.

3 Related Work

Point Cloud Management and Analysis. PC data has recently become a popular data type for analysis, driven by the rapid development of autonomous driving. Several studies [13, 18, 35, 40, 43] have focused on improving the prediction accuracy of deep models for tasks such as 3D classification [35, 41], object detection [39, 40, 49], and semantic segmentation [18, 36]. However, these works focus solely on improving model accuracy and do not address efficiency issues related to data storage and analysis. C_2O [29] performs efficient similarity search between a query PC object and candidate PC objects. MAST [26, 27] aims to improve the efficiency of analytical query processing on PC data by selectively sampling important PC frames for deep model inference, rather than processing all frames. However, it does not optimize disk consumption, as PC frames are still stored in raw format. Commercial database systems such as pgPointCloud [32] and Ganos [48] support storing PC data by treating each point cloud as an independent spatial object. These systems typically apply compression methods, such as LASzip [20] and dimensional compression, to individual frames without considering redundancy across frames. In this paper, we propose a method that jointly manages multiple *consecutive* PC frames and performs redundant point deduplication to achieve additional disk savings beyond existing storage compression approaches.

Point Cloud Storage Optimization. To reduce storage consumption when managing PC data, various compression methods have been developed. Octree-based approaches [8, 21, 23, 33] compress PC data by encoding it into Octree structures. Kammerl et al. [23] propose merging Octrees constructed from two consecutive PC frames to reduce temporal redundancy by identifying and combining identical subtrees. Other works [1, 16, 37] employ encoder-decoder architectures to compress and decompress Octree nodes, achieving additional compression gains. k^3-lidar [25] introduces a compact Octree-like indexing structure and stores additional point features using a columnar format. LASzip [20], a compression method specifically designed for PC data, applies techniques such as entropy coding and delta encoding to achieve efficient storage.

Our method focuses on redundancy elimination on a sequence of PC frames and is complementary to existing compression techniques. Experimental results also demonstrate that it can be effectively combined with compression methods like LASzip to achieve further reduction in storage consumption.

Data Redundancy Deduplication. The concept of deduplication [47] has been applied in managing other types of continuously generated unstructured data, such as video. For example, Haynes et al. [14] propose Video Storage System (VSS) [14, 15, 28], which optimizes video storage by identifying redundant image cells across multiple videos with overlapping spatial regions. [19] splits each video frame into fixed sized blocks by grid cells and jointly manage the blocks with high similarity. However, unlike images, which are structured as dense pixel grids, PC data consists of irregular, unordered sets of 3D points. This irregularity poses challenges for redundancy detection, but the rich spatial information of PC data also offers unique opportunities for geometry-aware deduplication. In this paper, we present the first framework that supports spatio-temporal deduplication across multiple consecutive PC frames, achieving up to 42% reduction in redundant points, as demonstrated in our experimental study.

4 Framework Overview

Design Challenges. We first introduce the key challenges when developing DUSTER, and how we address them.

Multi-PC Frame Joint Management. As introduced in Section 1, redundancy can occur across multiple neighboring PC frames, rather than being confined to adjacent pairs as assumed in [23]. It may also appear in both relative and absolute coordinate systems. To address this, DUSTER supports deduplication over an arbitrary number of consecutive PC frames and across both coordinate systems. Specifically, it segments the PC sequence into frame groups for joint deduplication, and applies an effective and efficient algorithm to generate shared PC point subsets (referred to as PC patches).

Budget Constraint Deduplication. In practice, the resource budget (e.g., CPU time) available for storage optimization and data preprocessing is typically limited [2], especially in scenarios such as PC data, where massive volumes of data continuously arrive. Given such constraints, finding a preferred deduplication strategy becomes non-trivial. To address this challenge, we propose a segmentation method that takes a resource budget as input. By leveraging integer programming and multi-armed bandit modeling, our approach efficiently derives an optimized deduplication plan under the resource constraint.

Efficient Data Loading. The framework is designed to both reduce storage usage and simplify data loading during retrieval. Existing approaches such as the Octree-based method [23] reduce redundancy by merging Octrees constructed from two consecutive PC frames. However, when performing random PC frame retrieval, this method requires loading the entire merged Octree to reconstruct the target frame. As shown in our experimental results (Section 6), the I/O cost of Octree loading is comparable to that of loading the raw data. To tackle this, we design a framework that ensures only the necessary shared PC point subsets are loaded, thereby significantly reducing I/O cost.

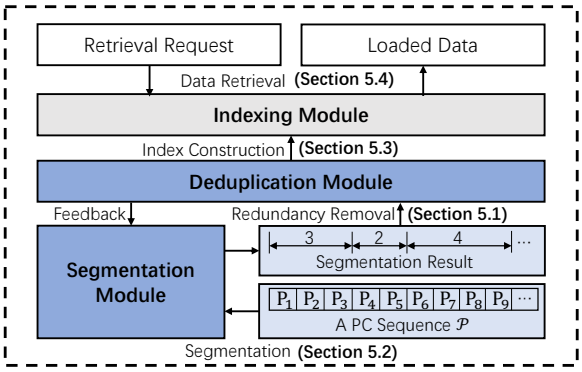


Figure 2: The framework of DUSTER.

Overview of DUSTER. The main architecture of DUSTER is provided in Fig. 2. The framework is mainly consisted of three modules,

namely (1) the *segmentation module* that partition the PC sequence $\mathcal{P}$ into segments, (2) the *deduplication module* that removes redundancy of each segment, and (3) the *indexing module* that processes efficient PC data saving and loading.

Specifically, when the PC sequence $\mathcal{P}$ arrives, the segmentation module first generates a preferred segmentation plan, where the PC frames in each segment are considered for joint optimization. Once the plan is determined, the deduplication module removes redundancy within each segment. The indexing module then indexes and manages the deduplicated data in a compact manner. During data retrieval, the indexing module loads and reconstructs the required data depending on the retrieval type (random or sequential).

5 Methodology

We proceed to introduce the methodology of DUSTER. We first describe the deduplication module (Section 5.1), followed by the segmentation module (Section 5.2), and the indexing module together with the data retrieval pipeline (Section 5.3).

5.1 Multi-Frame Joint Deduplication

Consecutive PC frames often exhibit spatio-temporal redundancy, and removing such redundancy is essential for storage optimization. We formally define redundant PC points and redundant PC cluster:

Redundant PC Points. Consider a set of N consecutive PC frames $\mathcal{P}_o = \{\mathsf{P}_1, \ldots, \mathsf{P}_N\}$, $|\mathcal{P}_o| = N$, jointly processed for deduplication. For a point p_1 in a PC frame $\mathsf{P}_i \in \mathcal{P}_o$, if there exists a point p_2 in $\mathsf{P}_j \in \mathcal{P}_o$ $(i \neq j)$ such that their Euclidean distance satisfies $d(p_1, p_2) \leq \theta$, where θ is a predefined distance threshold, then p_1 and p_2 are defined as two redundant PC points.

Redundant PC Cluster. Given $\mathcal{P}_o$, a redundant PC cluster is defined as a subset of points $c = \{p_{i_1}, p_{i_2}, \ldots, p_{i_k}\}$ satisfying the following two constraints:

(1) Uniqueness constraint: The cluster c contains at most one point from each PC frame, i.e., for any two points $p_1, p_2 \in c$, if $p_1 \in \mathsf{P}_i$, then $p_2 \notin \mathsf{P}_i$.
(2) Radius constraint: There exists a center q such that $d(p, q) \leq \theta$ for all $p \in c$.

The radius constraint means there exists a ball of radius θ that encloses c, whose center point q is considered as redundant PC point to all points $p \in c$. For each c identified from N PC frames, we only store the coordinate of q that represents the same spatial location within threshold θ, that is, the distance threshold $d(q, p) \leq \theta$ satisfied for all $p \in c$. Based on this concept, we now formulate the optimal PC deduplication problem, which aims to find a set of redundant PC clusters whose union covers all points across the N PC frames. Minimizing the number of clusters directly minimizes stored spatial coordinates, thus maximizing the deduplication performance.

Multi-PC Frame Redundancy Deduplication. We define the optimal PC deduplication problem on $\mathcal{P}_o$.

Problem 1 (Optimal PC Deduplication). *Given a set $\mathcal{P}_o$ of N point clouds, assume each point cloud contains at most M points. Let $d(p_1, p_2)$ denote the Euclidean distance between two points p_1 and p_2. A PC redundant cluster c is a subset of points from the union of all point clouds $\bigcup_{P \in \mathcal{P}_o} P$ that satisfies the previously defined uniqueness and radius constraints under the threshold θ.*

The problem aims to find a set of redundant PC clusters C^* with the minimum cardinality: $C^* = \operatorname{argmin}_{C \in \mathbb{C}} |C|$, where $\mathbb{C}$ denotes the domain of feasible cluster sets that satisfy the following conditions:

(1) Completeness: all points in $\mathcal{P}_o$ are contained in the union of all $c \in C$: $\bigcup_{c \in C} c = \bigcup_{\mathsf{P} \in \mathcal{P}_o} \mathsf{P}$;
(2) Disjointness: clusters are non-overlapping, i.e., $\forall c_1, c_2 \in C$, if $c_1 \neq c_2$, then $c_1 \cap c_2 = \emptyset$.

The cardinality of result C equals to the number of spatial coordinates that need to be stored. For example, a redundant PC cluster containing 3 points means that 2 redundant PC points are removed. If $N = 2$, the problem reduces to find a maximal bipartite matching of two sets and can be solved in polynomial time. Under the situation of $N \geq 3$, however, producing C^* becomes non-trivial. We prove that the problem is NP-hard when $N \geq 3$.

Theorem 5.1 (NP-hardness). *The optimal PC deduplication problem with $N \geq 3$ is NP-hard.*

Proof. (1) problem: Given N PC frames $F_1, F_2, \ldots, F_N$, $N \geq 3$, each containing a set of points in Euclidean space, an integer k, and a distance threshold D, the decision version of the PC frame deduplication problem is defined as: *Is there a partition of all points from the N frames into at most k clusters, such that each cluster contains at most one point from each frame and and each cluster is contained in a ball of radius D?*

(2) To prove the NP-hardness of this decision problem, we reduce the following *Colorful k-Center* (CKC) problem, which is known to be NP-complete [3, 11]. Given a set of points P in Euclidean space partitioned into color classes $\{C_1, C_2, \ldots, C_n\}$, an integer k, and a radius threshold D, the CKC problem asks: *Can the points in P be partitioned into at most k clusters, such that each cluster contains at most one point from each color class, and each cluster is contained in a ball of radius D?*

(3) We construct a reduction from the CKC problem to PC Frame Deduplication step by step. Given an instance of CKC: (i) Points P partitioned into n color classes $C_1, C_2, \ldots, C_n$, and (ii) integer k and radius threshold D. We construct an instance of PC frame deduplication as follows:

- Frame Construction: Create exactly $N = n$ frames. For each color class C_i, frame F_i contains exactly the points from C_i.
- Threshold Setting: Use the same radius threshold D.

This construction is clearly polynomial-time, as it simply involves copying points and assigning frames accordingly.

The equivalence between the two problems is straightforward. The set of clusters in the CKC problem can be converted to and from a set of clusters of the constructed deduplication problem. The constraint that "at most one point per frame" from PC frame deduplication problem exactly matches the "at most one point per color" constraint from the colorful k-center problem. This equivalence implies that the PC Frame Deduplication decision problem, as well as its optimization version (minimizing the number of clusters given a radius constraint) are both NP-hard.

□

Deduplication Operation. Beyond its computational hardness, the problem becomes more complex when redundancy in both the

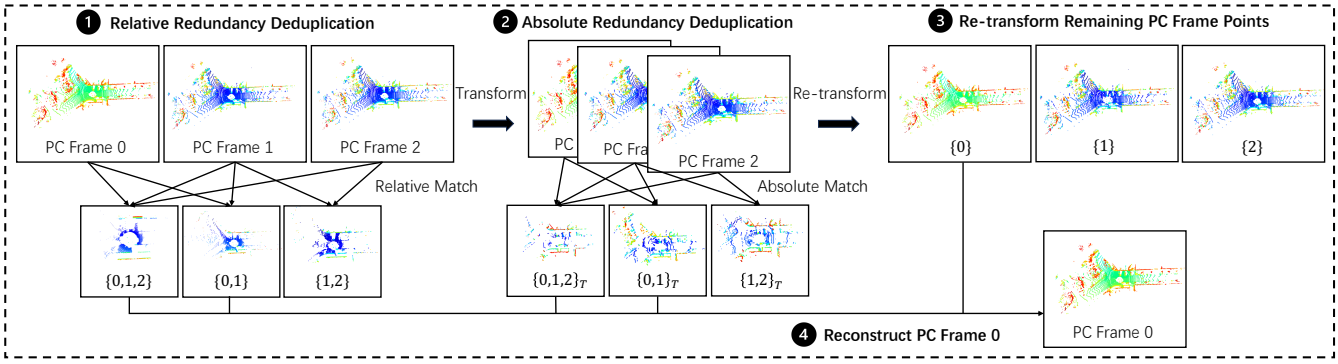


Figure 3: Example of the PC frame deduplication and reconstruction process.

relative and absolute coordinate systems is considered. Therefore, we aim to develop a method that effectively detects redundant PC clusters while maintaining processing efficiency. We proceed to introduce an effective and efficient deduplication operation $\mathrm{De}(\mathcal{P}_o, \theta)$, where θ denotes the distance threshold.

PC Frame Combination Generation. Given $\mathcal{P}_o$ of N PC frames, the algorithm first generates all possible combinations of the N frames, from size N to size 2. Specifically, there are C_N^n distinct combinations of size $n \in [2, N]$. Then, the algorithm performs redundant PC cluster detection for each combination, aiming to identify the maximum number of redundant PC clusters, where each cluster contains exact one point from each PC frame in the combination and satisfies the disjointness constraint described in Problem 1. For example, when considering a combination of two frames $\{\mathsf{P}_1, \mathsf{P}_2\}$, the algorithm identifies redundant PC clusters between P_1 and P_2, where each cluster c consists of two points (one from each frame).

Process Workflow. After the generation of frame combinations, the algorithm proceeds as follows. It begins by detecting redundant clusters from combinations of larger sizes and then proceeds to smaller ones; that is, it first detects redundant PC clusters of size N. This strategy is adopted because a cluster of size N provides the maximum deduplication benefit, eliminating $N-1$ redundant PC points. Maximizing the number of larger clusters consequently reduces the total number of clusters in the final result. It then proceeds to detect redundant PC clusters for combinations of size $N-1$, continuing until all combinations of size 2 have been processed.

After performing cluster detection on a combination, the matched points (i.e., points contained in any detected redundant PC cluster c) are removed from the point sets of their corresponding PC frames. This prevents subsequent detections from violating the disjointness constraint. We denote redundant PC cluster detection on a combination of n PC frames as an *n-frame detection*. The resulting matched clusters form a set C^p referred to as a *PC Patch*, defined as follows:

Definition 5.2 (PC Patch). A PC patch C^p is defined as a set of redundant PC clusters. $\forall c \in C^p, |c| = n$, where n is the number of PC frames. An n-frame detection produces a PC patch containing redundant PC clusters with cardinality of n.

Particularly, empirical observations indicate that non-consecutive frame combinations (e.g., combination of $\{\mathsf{P}_1, \mathsf{P}_3\}$, where the frame indices are not consecutive), typically do not exhibit sufficient redundancy. Therefore, our method excludes these combinations from consideration. After all detections, any remaining unmatched points of each PC frame are treated as singleton clusters containing only one point, and forms one PC patch for each frame.

An example of the deduplication procedure is given as follows.

Example 5.3. As shown in the step 1 of Fig. 3, given three PC frames $\{0, 1, 2\}$, the algorithm first detects redundant PC clusters of size 3. After completing the 3-frame detection, the algorithm proceeds to detect redundancy across smaller subsets of frames, specifically considering 2-frame detection. In this step, only combinations of consecutive frames, namely $\{0, 1\}$ and $\{1, 2\}$, are evaluated. Combinations such as $\{0, 2\}$ are excluded from consideration since they are non-consecutive.

Redundant PC Cluster Detection. We now describe how the n-frame detection is performed for each combination of size n with threshold θ. Given a subset of n PC frames $\mathcal{P}_o^n \subseteq \mathcal{P}_o$, the algorithm selects one PC frame as the anchor frame for matching redundant PC points. This transforms the detection of redundant PC clusters across n PC frames into a series of redundant PC point matchings between the anchor frame and each of the remaining $n-1$ frames, thereby reducing the detection complexity. Specifically, The PC frames are first sorted by their generation timestamps. If n is odd, the middle frame is chosen as the anchor; if n is even, the leftmost of the two central frames (i.e., the $(n/2-1)$ th frame) is selected as the anchor. We select the middle frame as the anchor since it serves as a balanced representative among all frames and typically exhibits the greatest overlap with other frames, enabling the detection of more redundant PC clusters.

The algorithm then performs redundant PC point matching between the anchor frame P_a and each of the remaining $n-1$ frames $\mathsf{P}_r \in \mathcal{P}_o^n \setminus \mathsf{P}_a$. Specifically, with P_a and P_r, the algorithm takes every point $p_a \in \mathsf{P}_a$ as a query point and conducts k nearest neighbor (kNN) search (we set $k = 10$ by default, clustering results are insensitive to k) on the points of P_r and returns k nearest points $k\mathrm{NN}(p_a, \mathsf{P}_r) \subseteq \mathsf{P}_r$, where each $p_r \in k\mathrm{NN}(p_a, \mathsf{P}_r)$ satisfies the distance constraint $d(p_a, p_r) \leq \theta$. In other words, the kNN operator returns the k nearest redundant PC points of p_a. If no point in P_r satisfies the constraint, kNN returns an empty set $\emptyset$. After performing all matching across all frames in $\mathcal{P}_o^n \setminus \mathsf{P}_a$, the points in P_a are sorted according to their average nearest-neighbor distance: $d_a(p_a) = \frac{\sum_{\mathsf{P}_r \in \mathcal{P}_o^n \setminus \mathsf{P}_a} \left(\min_{p_r \in k\mathrm{NN}(p_a, \mathsf{P}_r)} d(p_a, p_r) \right)}{|\mathcal{P}_o^n \setminus \mathsf{P}_a|}$, where

$\min_{p_r \in k\text{NN}(p_a, \mathsf{P}_r)} d(p_a, p_r)$ denotes the distance from p_a to the top-1 nearest neighbor in P_r. The redundant PC clusters are then constructed following the sorted order of P_a. Starting from the point p_a with the smallest $d_a(p_a)$, the algorithm assigns the nearest unselected neighbor from each frame in $\mathcal{P}_o^n \backslash \mathsf{P}_a$. If any frame $\mathsf{P}_r \in \mathcal{P}_o^n \backslash \mathsf{P}_a$ yields $k\text{NN}(p_a, \mathsf{P}_r) = \emptyset$, the algorithm skips to the next point. Otherwise, p_a together with its $n-1$ assigned points each from one frame in $\mathcal{P}_o^n \setminus \mathsf{P}_a$ form a redundant PC cluster that can be covered by a ball of radius θ with p_a as the center. The algorithm stops when the point p_a is with $d_a(p_a) > \theta$, which means not satisfying distance constraint anymore. To efficiently handle the matching process, we leverage Faiss [10] for batched kNN search computations.

After the detection, a set of redundant PC clusters with radius θ are generated, forming a PC patch. We then only stores the spatial coordinate of each p_a that represents the coordinate of a redundant PC cluster, since the distance between coordinate of p_a and the original point is bounded by θ.

Algorithm 1: Deduplication Operation $\text{De}(\mathcal{P}_o, \theta)$.

input : N frames of PC frames $\mathcal{P}_o$, the distance threshold θ;
output : Deduplicated PC point Patches $\mathcal{P}_d$;

1 $\mathcal{S} \leftarrow$ clustering PC frame combinations generated by $\mathcal{P}_o$;
2 **for** $\mathcal{P}_o^n \in \mathcal{S}$ **do**
3 $C^p = \emptyset$;
4 $\mathsf{P}_a \leftarrow \text{Init}(s, \mathcal{P}_o^n)$;
5 $M_s \leftarrow \{\emptyset \text{ for } \_ \text{ in range}(|\mathsf{P}_a|)\}$;
6 **for** $\mathsf{P}_r \in \mathcal{P}_o^n \setminus \mathsf{P}_a$ **do**
7 $I_m \leftarrow \text{match}(\mathsf{P}_a, \mathsf{P}_r)$;
8 **for** $i_a, i_p \in I_m$ **do**
9 $M_s[i_a].\text{append}(i_p)$;
10 sort(M_s) ;
11 **for** $i_a, m \in \text{enumerate}(M_s)$ **do**
12 **if** $|m| = |\mathcal{P}_o^n \setminus \mathsf{P}_a|$ **then**
13 $p = (\mathsf{P}_a[i_a].xyz, [\mathsf{P}_a[i_a].int])$;
14 **for** $\mathsf{P}_r \in \mathcal{P}_o^n \setminus \mathsf{P}_a$ **do**
15 $i_\mathsf{P} = m(\mathsf{P}_r)$; $p[1].append(\mathsf{P}[i_\mathsf{P}].int)$;
16 P. delete(i_P);
17 C^p. append(p);
18 P_a. delete(i_a);
19 $\mathcal{P}_d.append(C^p)$;
20 **return** $\mathcal{P}_d$

The pseudo code of $\text{De}(\mathcal{P}_o, \theta)$ is presented in Alg. 1. It first generates a list of clustering combinations $\mathcal{S}$ (line 1). For combination $\mathcal{P}_o^n$, e.g., $\{1, \ldots, N\}$, a deduplicated PC patch C^p is initialized as empty (line 3). The anchor PC frame P_a is then initialized (line 4). A match list M_s is also initialized to store the matching results between points in P_a and those in $\mathcal{P}_o^n \setminus \mathsf{P}_a$ (line 5). For each frame P_r in $\mathcal{P}_o^n \setminus \mathsf{P}_a$, a matching is performed (line 7) to detect redundant PC points between P_a and P_r, which are stored in M_s (lines 8–9). The points in M_s are sorted w.r.t. $d_a(p_{i_a})$. Next, the satisfied cases in M_s (i.e., those where there exists one matched point from each $\mathsf{P}_r \in \mathcal{P}_o^n \setminus \mathsf{P}_a$ (line 12)) are clustered (lines 13–17). Only the spatial coordinate of the anchor point $\mathsf{P}_a[i_a].xyz$ is stored (line 13), while all light intensity features are retained (line 15). Finally, the matched points are deleted from the PC frames (lines 16 and 18).

Deduplication Operator Time Complexity. The main computational cost of the deduplication process lies in performing matching between pairs of PC frames. Assuming that the number of points in each PC frame is bounded by $|\mathsf{P}|$, the time complexity for matching between two PC frames is $O(|\mathsf{P}| \log |\mathsf{P}|)$ [6, 10]. For deduplication across N PC frames, without matching result reuse, the total number of matchings required is $\sum_{i=0}^{N-2} (N-i-1)(i+1) = \frac{N}{6}(N^2-1)$, which is $O(N^3)$. This is because, for each index i, there are at most $i+1$ possible satisfied combinations to consider, and each requires $N-i-1$ matchings. Therefore, the overall time complexity of the deduplication process is: $O(N^3 \cdot |\mathsf{P}| \log |\mathsf{P}|)$. This worst-case bound occurs when no matched points are removed and the point count $|\mathsf{P}|$ remains constant throughout.

Soft Distance Threshold Matching. To improve the deduplication performance, we relax the strict point-wise distance constraint $d(p_1, p_2) \leq \theta$ used during matching between two PC frames. Specifically, we introduce a *soft distance threshold* mechanism that combines a relaxed point-wise bound with an additional set-wise average distance constraint. Specifically, for the matching result $\mathcal{P}_m = \{(p_1^1, p_1^2), \ldots, (p_n^1, p_n^2)\}$ returned by the algorithm, the following conditions must hold:

$$\forall (p, q) \in \mathcal{P}_m,\ \text{dist}(p, q) < \Theta, \tag{1}$$

$$\frac{\sum_{(p,q) \in \mathcal{P}_m} \text{dist}(p, q)}{|\mathcal{P}_m|} < \theta, \tag{2}$$

where Θ is a relaxed upper bound defined as $\Theta = \alpha \cdot \theta$ with $\alpha > 1$ (default is 2). This soft thresholding approach allows individual matched pairs to exceed θ as long as the *average matching distance* across the entire set remains below θ, providing flexibility in the matching process while controlling overall redundancy quality. This mechanism is evaluated in Section 6.2.

Multi-frame & Multi-coordinate system Deduplication. We further extend the deduplication algorithm to address redundancy in both the relative and absolute coordinate systems, as redundancy is observed in both (as shown in Fig. 1). Given $\mathcal{P}_o$ with points of each PC frame in the relative coordinate system, the extended algorithm is as follows: we first produce the relative redundancy $\text{De}(\mathcal{P}_o, \theta)$ and remove the matched redundant points. The remaining points in each PC frame are then transformed into the absolute coordinate system by applying a 6-DOF pose transformation [42], i.e., $\mathsf{P}^t = \text{Transform}(\mathsf{P}, pose)$. Then, we compute the absolute redundancy $\text{De}(\mathcal{P}_o^t, \theta)$, where $\mathcal{P}_o^t$ denotes the transformed PC frames.

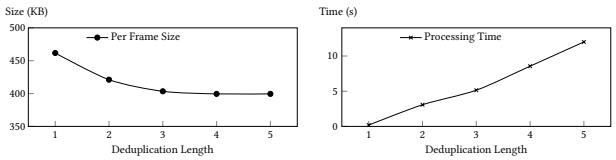
For a set of N PC frames, the pseudo code of a complete deduplication procedure based on $\text{De}(\cdot, \cdot)$ is presented in Alg. 2. The algorithm first performs deduplication on the input PC frames $\mathcal{P}_o$ in the relative coordinate system, where the frames are originally stored (lines 1–2). The unmatched points are then transformed into the absolute coordinate system (line 3) and further deduplicated under this system (lines 4–5). The coordinates are re-transformed back to relative coordinate systems after deduplication (line 6). Finally, the matched results from both coordinate systems are merged and returned (line 7).

Algorithm 2: Multi-frame & coordinate system Deduplication Procedure.

input : N frames of PC frames $\mathcal{P}_o$, the distance threshold θ, transpose poses $poses$;
output : Deduplicated PC point patches $\mathcal{P}_d$;

$\mathcal{P}_d^1 \leftarrow \text{De}(\mathcal{P}_o, \theta)$
$\mathcal{P}_o.\text{delete}(\mathcal{P}_d^1)$
$\mathcal{P}_o^t \leftarrow \text{Transform}(\mathcal{P}_o, poses)$
$\mathcal{P}_d^2 \leftarrow \text{De}(\mathcal{P}_o^t, \theta)$
$\mathcal{P}_o.delete(\mathcal{P}_d^2)$
$\mathcal{P}_o, \mathcal{P}_d^2 \leftarrow \text{Re-transform}(\mathcal{P}_o, \mathcal{P}_d^2)$
$\mathcal{P}_d \leftarrow \mathcal{P}_d^1 + \mathcal{P}_d^2 + \mathcal{P}_o$
return $\mathcal{P}_d$

Example 5.4. Fig. 3 gives an example of the whole deduplication. Three PC frames are selected to be deduplicated. In step 1, the algorithm deduplicates the PC frames in the relative coordinate system by Alg. 1, and results in three PC patches: $C^{\mathcal{P}}_{\{0,1,2\}}$, $C^{\mathcal{P}}_{\{0,1\}}$, and $C^{\mathcal{P}}_{\{1,2\}}$. Then, the system rotates and translates the remained points in each PC frame into the absolute coordinate system based on the *pose* of each frame. Alg. 1 is then conducted on the transformed points (step 2), which also results in 3 PC patches: $C^{\mathcal{P}}_{\{0,1,2\}_T}$, $C^{\mathcal{P}}_{\{0,1\}_T}$, and $C^{\mathcal{P}}_{\{1,2\}_T}$, where T indicates that the patches are constructed under a transformed situation. Finally, the remained points are transformed back to the relative coordinate systems, as three new PC patches, namely $C^{\mathcal{P}}_{\{0\}}$, $C^{\mathcal{P}}_{\{1\}}$, and $C^{\mathcal{P}}_{\{2\}}$.



(a) Per frame file size. **(b) Per frame processing time.**

Figure 4: Example showing how the average deduplication performance varies with the deduplication frame length on the SemanticKITTI [5] dataset.

5.2 PC Sequence Segmentation

The computational complexity of the deduplication algorithm increases as the number of PC frames N grows, and the performance gain saturates when N becomes large. As shown in Fig. 4, the average size reduction of each PC frame, when combined with a compression method, becomes marginal once the deduplication frame length reaches 5. Therefore, applying the deduplication algorithm jointly on all PC frames of a sequence is both ineffective and inefficient. To address this, we design a segmentation module that partitions the PC sequence $\mathcal{P}$ into segments $Seg = \{seg_1, \ldots, seg_s\}$, each containing a consecutive subsequence of PC frames. Deduplication is then performed independently on each segment. This approach enables efficient redundancy removal while maintaining desirable deduplication performance. Furthermore, since the overlap region between consecutive PC frames can vary over time, uniformly splitting the sequence may miss opportunities to achieve higher deduplication performance under a fixed computation budget. To tackle this, we develop a novel segmentation algorithm that adaptively partitions the PC sequence within the computation budget. The segmentation procedure is formally defined as follows:

Definition 5.5 (PC Sequence Segmentation). Given a PC sequence $\mathcal{P}$, the segmentation module generates a segmentation proposal $Seg = \{seg_1, \ldots, seg_s\}$, where each segment seg_i contains consecutive PC frames (sorted by timestamp). The union of all segments covers the entire sequence, i.e., $\bigcup_{seg_i \in Seg} \{\mathsf{P} \mid \mathsf{P} \in seg_i\} = \mathcal{P}$. The objective of the segmentation is to minimize the total disk space usage after deduplication:

$$Seg = \arg\min_{Seg_i} \sum_{seg \in Seg_i} \text{size}(seg), \tag{3}$$

where size$(\cdot)$ returns the file size of a segment after deduplication.

In practical scenarios, storage optimization operates under limited computational resources. Given a CPU time budget B, the optimization objective becomes the Eq. 3 subject to the constraint: $\sum_{seg \in Seg_i} \text{time}(seg) \le B$, where time$(seg)$ denotes the processing time required for deduplicating segment seg.

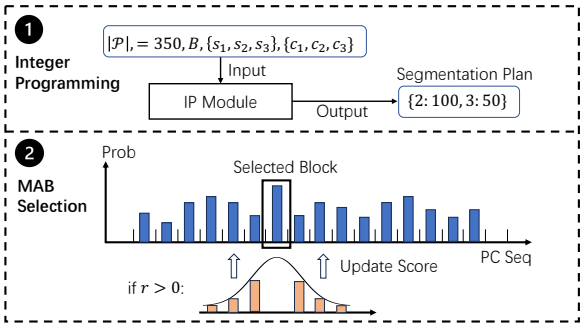


Figure 5: An example of PC sequence segmentation.

Segmentation via Integer Programming and Bandit-based Refinement. Our segmentation method first leverages integer programming [46] to globally determine the number of segments of each length, and then applies a multi-armed bandit (MAB) algorithm [22] with local Gaussian smoothing to adaptively assign the longer segments to the most beneficial temporal regions.

(1) Integer Programming Modeling with Time Budget. We begin by formulating the segmentation problem as an integer programming task that considers the estimated time cost and resulting PC frame size for each possible segment length. Let x_ℓ denote the number of segments of length $\ell \in \mathbb{Z}_{\ge 0}$, c_ℓ be the estimated processing time of such a segment, and s_ℓ be its expected compressed file size (or point set size after deduplication, depending on the optimization objective). Given a total number of frames $|\mathcal{P}|$ and a time budget B, we formulate the following integer program:

$$\text{Minimize:} \quad \sum_{\ell=1}^{L} x_\ell \cdot s_\ell, \tag{4}$$

$$\text{Subject to:} \quad \sum_{\ell=1}^{L} x_\ell \cdot \ell = |\mathcal{P}|, \quad \sum_{\ell=1}^{L} x_\ell \cdot c_\ell \le B,$$

$$x_\ell \in \mathbb{Z}_{\geq 0}, \quad \forall \ell \in \{1, \ldots, L\}; \tag{5}$$

This formulation guarantees full coverage of the input sequence (equality constraint on the total number of frames), adherence to the time budget, and optimal segmentation with respect to the modeled compression performance.

In practice, we observe that due to the piecewise convex nature of s_ℓ and c_ℓ, the optimal solution typically consists of only two adjacent segment lengths (e.g., ℓ and $\ell + 1$), possibly with a few additional segments of length 1 to satisfy the constraint $\sum_{\ell=1}^{L} x_\ell \cdot \ell = |\mathcal{P}|$. For example, if the average time budget per frame lies between $c_2/2$ (the per-frame cost for 2-frame segments) and $c_3/3$ (for 3-frame segments), the solution tends to include only segments of length 2 and 3.

As illustrated in the upper part of Fig. 5, the algorithm sets $N_M = 3$, with the expected file sizes for each segment length given by $\{s_1, s_2, s_3\}$ and the corresponding deduplication processing times by $\{c_1, c_2, c_3\}$. The integer programming module determines that the optimal assignment includes 100 2-frame segments and 50 3-frame segments, which together partition the sequence $\mathcal{P}$ of size $|\mathcal{P}| = 100 \times 2 + 50 \times 3 = 350$.

We observe that, because of the spatio-temporal continuity among consecutive PC frames, segments with strong deduplication performance tend to have neighboring segments with similarly good performance, and vice versa. Motivated by this observation, we design our segmentation assignment algorithm as follows:

(2) Multi-Arm Bandit Selection with Gaussian Smoothing. Once the segment length assignment is determined, we refine the placement of longer segments using a bandit-based approach. Specifically, we partition the input sequence $\mathcal{P}$ into non-overlapping blocks of length equal to the least common multiple (LCM) of the selected segment lengths (e.g., $\text{lcm}(\ell, \ell + 1)$). This ensures that each block can be evenly divided into segments of both lengths ℓ and $\ell + 1$. Each block is treated as an arm in the MAB problem, where pulling an arm corresponds to assigning it a group of longer segments.

The scores of all arms are initialized to 1. At each selection step, the algorithm randomly selects an arm from the set of unselected arms according to the selection probability: $\text{Prob}(i) = \frac{\text{score}(i)}{\sum_{j \in \mathcal{B}} \text{score}(j)}$, where $\mathcal{B}$ denotes the set of partitioned blocks, and $\text{score}(i)$ is the assigned score of arm i.

After selecting a block to assign a longer segment, we estimate its reward by measuring the improvement in deduplication performance achieved when using longer segments instead of shorter ones.

$$pg = \sum_{seg_1 \in \text{Seg}_1} \text{size}(seg_1) - \sum_{seg_{\ell+1} \in \text{Seg}_{\ell+1}} \text{size}(seg_{\ell+1}), \tag{6}$$

where $\text{Seg}_{\ell+1}$ (resp. Seg_1) denotes the segmentation result of the selected LCM block into segments of length $\ell + 1$ (resp. 1). Here, we assign segments of length 1 as short segments, since computing their size incurs relatively low overhead compared to the deduplication process on longer segments (as shown in Fig. 4), thus introducing only minimal extra cost when calculating the reward.

We maintain pg_a, the average performance gain observed from previously selected blocks. The reward is then defined as $r = (pg - pg_a)/pg_a$. A positive reward indicates that the current block's performance gain exceeds the historical average, while a negative reward indicates it falls short of expectations.

After selecting arm i, we update the scores of its neighboring unselected arms using a Gaussian-smoothed adjustment [44] weighted by the signed reward:

$$\Delta_{\text{score}}(r, j) = \eta \cdot r \cdot \exp\left(-\frac{(j-i)^2}{2\sigma^2}\right), \quad \forall j \in \mathcal{N}(i). \tag{7}$$

Here, r denotes the normalized performance gain for arm i, η is the learning rate, and σ is the exponential decay coefficient (with a default value of 2). $\mathcal{N}(i)$ denotes the 6 nearest neighboring arms (three on each side of i; e.g., if $i = 3$, then $\mathcal{N}(i) = \{0, 1, 2, 4, 5, 6\}$). This localized Gaussian-smoothed update mechanism promotes spatial coherence in segment assignment—reinforcing high-performing regions for future sampling while down-weighting poorly performing ones.

As shown in the bottom part of Fig. 5, after a block is selected, if the reward r is positive, the algorithm increases the scores of its neighboring blocks, thereby increasing their probabilities of being selected in future arm selections.

Remaining unselected blocks and leftover frames are filled using the shorter segment length. The final segmentation ensures complete coverage, balanced processing time, and optimized deduplication performance under budget constraints. Further, in practice, arm selections are performed in batches—i.e., n blocks are selected before the selection distribution is updated using Eq. 7. This batching strategy improves the efficiency of the segmentation procedure and stabilizes the estimate of pg_a.

Algorithm 3: IP-MAB Segmentation Algorithm.

input : PC sequence $\mathcal{P}$, Expected processing time C and expected size S, time budget B;
output : Segmentation Proposal Seg;

1 `Segment_Assignment` $\leftarrow \text{IP}(\mathcal{P}, B, C, S)$
2 $\mathcal{B}, \mathcal{P}_l \leftarrow$ Set_Block($\mathcal{P}$, `Segment_Assignment`)
3 $score(b_i) \leftarrow 1$ for $\forall b_i \in \mathcal{B}$
4 $N_{\ell+1}$ denotes the number of blocks assigned longer segments of length $\ell + 1$
5 Initialize $pg_a = 0$, segmentation result $Seg \leftarrow \emptyset$
6 **while** $N_{\ell+1} > 0$ **do**
7 $b_i \leftarrow$ select($score, \mathcal{B}$)
8 Seg.append(segments of length $\ell + 1$ split by b_i)
9 Compute pg according to Eq. 6
10 $r \leftarrow (pg - pg_a)/pg_a$
11 **for** $j \in \mathcal{N}_i$ **do**
12 $score(j) \leftarrow score(j) + \Delta_{score}(r, j)$
13 Update pg_a
14 $N_{\ell+1} \leftarrow N_{\ell+1} - 1$
15 Assign the unselected blocks and $\mathcal{P}_l$ with the shorter segments
16 **return** Seg

The segmentation procedure is summarized in Algorithm 3. Given the time budget B, the estimated processing costs $C = \{c_1, c_2, \ldots, c_{N_M}\}$, and the corresponding output sizes $S = \{s_1, s_2, \ldots, s_{N_M}\}$, the Integer Programming module first determines the segment length assignment (line 1). Then, the PC sequence is partitioned

into a set of blocks $\mathcal{B}$ and remaining frames $\mathcal{P}_l$ (line 2). All block scores are initially set to 1 (line 3), and the average performance gain pg_a as well as the final segmentation list Seg are initialized (line 5). During each iteration of the loop (lines 6–14), a block b_i is selected according to the probability distribution defined over the scores (line 7). The frames in b_i are segmented with length $\ell + 1$ (line 8), and the performance gain pg and normalized reward r are computed (lines 9–10). The scores of unassigned neighboring blocks b_j of b_i are then updated using Gaussian smoothing (lines 11–12), and pg_a is updated accordingly (line 13). Finally, any remaining unassigned blocks are segmented into shorter segments (length less than $\ell + 1$) based on the segment assignment.

5.3 Compact Data Management and Indexing

Multi-frame Indexing and Saving. After each segment $seg = \mathcal{P}_o$ is deduplicated, the resulting PC patches $\mathcal{P}_d$ are organized by maintaining an index structure to support efficient PC frame retrieval. The index stores the metadata: (1) the poses of the PC frames, (2) the PC patches $\mathcal{P}_d$, (3) the anchor frame ID for each patch, and (4) the list of PC frames associated with each patch. E.g., $(\texttt{Patch}, \{1, 2, 3\}, 2)$ indicates that patch C^p is shared across PC frames {1, 2, 3}, with PC frame 2 as the anchor. During the retrieval process, when PC frame i is requested, the index locates the corresponding patches and reconstructs the frame accordingly.

The design of DUSTER is inherently orthogonal to existing compression techniques, as each C^p represents a subset of a point cloud frame and can therefore be directly compressed. We demonstrate this by integrating the *LASzip* method [20] to compress PC patches for compact storage. Specifically, for a PC patch deduplicated from three PC frames, where each cluster is represented as $\{(x, y, z), (F_1, F_2, F_3)\}$ and F_i denotes the additional features of the i-th PC frame, we apply LASzip compression to the spatial coordinates (x, y, z) and store them in a *row-oriented layout*. The feature data F_i are stored in a *column-oriented layout*, which enables selective loading of only the required F_i during retrieval. In the special case where a segment contains only a single PC frame, we directly compress and store the frame using LASzip.

Efficient Data Retrieval. We develop the retrieval algorithm to support both random and sequential PC frame retrieval. The index and PC patches are persisted to disk, and only the necessary data is loaded on demand during retrieval. For random PC frame retrieval, where a single PC frame $\mathtt{P}_i$ is requested for analysis, the framework first locates the corresponding index file $\mathtt{I}$. After locating $\mathtt{I}$, the system loads the required metadata. The algorithm then retrieves the PC patches associated with $\mathtt{P}_i$. If $\mathtt{P}_i$ is the anchor frame of a patch $\mathtt{Patch}_i$, the patch is directly used; otherwise, the patch is transformed into the relative coordinate system of $\mathtt{P}_i$ according to the stored poses. For sequential PC frame retrieval, where multiple consecutive frames are required, the algorithm loads all relevant deduplicated patches and reconstructs the targeted frames in a manner similar to random retrieval, applying the appropriate transformations based on the pose information.

As shown in Fig. 3, during the data retrieval period (step 3), Suppose the query requires to load the PC frame 0, the algorithm collects the related PC patches, including $C^p_{\{0,1,2\}}$, $C^p_{\{0,1\}}$, $C^p_{\{0,1,2\}_T}$, $C^p_{\{0,1\}_T}$, and $C^p_{\{0\}}$, and reconstructs the PC frames correspondingly.

PC Reconstruction Accuracy Analysis. With the deduplication method, each reconstructed point is either an original point or a mapped point from the anchor PC frame. For the original points, the error is 0, while for the points reconstructed from PC patches, the average distance between the mapped points and their original counterparts is constrained by θ, as defined in Eq. 2. Therefore, the Chamfer distance [4] between the original PC frame and the reconstructed PC frame is upper bounded by θ.

PC Frame Retrieval Time Complexity. To retrieve a PC frame, the framework loads the associated PC patches, with a time complexity linear to $O(|\mathtt{P}|)$, where $|\mathtt{P}|$ denotes the number of points in the target frame. For the patches that require coordinate transformation (rotation and translation), the transformation cost is also bounded by $O(|\mathtt{P}|)$. Therefore, the overall reconstruction time complexity for a PC frame is $O(|\mathtt{P}|)$. Empirical results show that our method takes less than 0.05 s to load and reconstruct a PC frame, which is comparable to the performance of the LASzip method.

6 Evaluation

The experimental study aims to answer the following questions: **RQ1.** How effective is DUSTER in deduplicating PC data, in terms of both deduplication point percentage and storage consumption? **RQ2.** How does DUSTER perform on random PC data retrieval and sequential PC data retrieval? **RQ3.** How well does DUSTER preserve deep model accuracy on reconstructed PC frames? **RQ4.** How does the proposed IP-MAB segmentation method perform under varying time budgets? **RQ5.** How does DUSTER perform with varying distance threshold? **RQ6.** What is the effect of performing deduplication only in the relative or absolute coordinate system?

6.1 Experimental Setup

We now describe the experimental setup of our evaluation. The key parameters are listed in Tbl 1. We then introduce the datasets, retrieval workloads, comparison methods, and evaluation metrics.

Table 1: Experimental parameters.

Parameter	Value
Dataset	**SemanticKITTI**, ONCE
Distance Threshold θ (m)	0.02, 0.025, **0.03**, 0.035, 0.04
Soft Distance Threshold Ratio α	1, 1.5, **2**, 2.5, 3

Dataset. We evaluate our method on the following two real datasets:

- **SemanticKITTI** [5]**.** The SemanticKITTI dataset consists of 22 PC sequences collected by a moving vehicle across various road environments. Each sequence contains several thousand frames, captured at 10 frame per second (FPS), with a 0.1-second interval between consecutive frames.
- **ONCE** [31]**.** The ONCE dataset contains a total of one million PC frames, organized into sequences captured at 2 FPS.

The points in both dataset contain light intensity as the additional feature. Each PC sequence in SemanticKITTI or ONCE ranges from 2.5 GB to 9.2 GB in size, corresponding to several hundred seconds of captured data. The FPS difference makes that SemanticKITTI has a more intense frame situation that two cointuous PC frames has smaller time gap (0.1 s), while ONCE has a more sparse situation that two continuous PC frames have larger time gap (0.5 s). We

conduct experiments on both dataset to evaluate the performance of our method in both intense and sparse situation.

Data Retrieval Workload. We generate the random and sequential PC frame retrieval workload following a uniform distribution in each sequence. Specifically, we generate 100 random retrievals each retrieves a PC frame, and 10 sequential retrievals each retrieves 30 consecutive PC frames to conduct I/O cost evaluation.

Comparison Methods. We evaluate the following method to answer the research questions.

- **Raw Storage (RS).** This baseline does not apply storage optimization method to the PC data. Data are stored in raw data directly, which simulates the main stream deep learning projects.
- **Octree** [23]**.** It applies an Octree structure do conduct PC data compression. It supports to jointly compress two consecutive PC frames and merge their Octree structures. The light intensity is stored in a column-store style, apart from the spatial coordinate following [25]. LASzip is applied to compress the intensity feature. The maximum depth of an Octree is set as 15.
- **LASzip** [20]**.** This method is dedicated designed to compress PC data, which supports to compress the point cloud with further features such as light intensity.
- **DUSTER.** Our proposed method that deduplicates the PC sequence. It applies LASzip to compress the PC patches. We also include DUSTER-SX, where X denotes that all PC frames are segmented into fixed-length segments of X frames, and deduplication is performed within each segment. We set the default computation budget in DUSTER such that half of the PC frames are segmented into 3-frame segments, while the remaining half are segmented into 2-frame segments, balancing computational cost and deduplication performance.

Metrics. We evaluate the methods with the following metrics:, (1) **Deduplicated Points Percentage.** It computes the percentage of deduplicated points: $\mathtt{DPP} = 1 - \sum_{\mathsf{P}_j \in \mathcal{P}_d}(|\mathsf{P}_j|)/\sum_{\mathsf{P}_i \in \mathcal{P}_o}(|\mathsf{P}_i|)$. (2) **Storage Consumption.** It measures the disk consumption size of the evaluated PC sequence $\mathcal{P}$ after optimized by each method. (3) **Retrieval I/O cost.** We evaluate the I/O cost (i.e., size of data loaded) for both random and sequential retrieval. (4) **Model Prediction Accuracy.** We evaluate the consistency of the deep model's predictions between the reconstructed and original point cloud frames to measure the reconstruction quality. We apply the 3D object detection model PV-RCNN [39, 43] (pretrained on SemanticKITTI dataset) as the deep model following [26].

Evaluation Platform. The experiments are conducted on an 80-core server with Intel(R) Xeon(R) Gold 6248 CPU@2.50GHz, 64GB RAM, and 2 NVIDIA GeForce RTX 2080 Ti Rev. A GPUs. We release our codebase[1].

6.2 Experimental Results

RQ1. Deduplication Performance Evaluation. We evaluate the deduplication performance of our methods, DUSTER-S2 and DUSTER compared to baselines, in terms of deduplication point percentage and storage consumption.

(1) Deduplication Percentage. The experimental results on deduplicated points percentage are reported in Tbl. 2. On the SemanticKITTI dataset, DUSTER-S2 achieves an average of 32.82% reduction in the number of points that need to be stored per PC sequence (ranging from 29.59% to 36.92%), while DUSTER achieves an average of 37.50% (ranging from 33.80% to 42.07%). On the ONCE dataset, DUSTER-S2 achieves an average reduction of 31.65% (ranging from 29.65% to 32.76%), and DUSTER achieves 37.58% (ranging from 34.71% to 39.15%). DUSTER consistently outperforms DUSTER-S2 in deduplication point percentage across all sequences in both datasets, as the additional 3-frame deduplication budget enables the detection and removal of more redundancy. Moreover, DUSTER achieves comparable performance on SemanticKITTI and ONCE, indicating robustness to differences in generation time gaps between neighboring PC frames (0.5 s for ONCE vs. 0.1 s for SemanticKITTI).

Table 2: Deduplicated points percentage of DUSTER-S2 and DUSTER across different PC sequences.

Dataset	|Seq|	Deduplicated Points Percentage (%)	
		DUSTER-S2	DUSTER
SemanticKITTI	4,541	31.39	**35.89**
	4,661	36.92	**42.07**
	4,071	30.62	**35.09**
	4,982	29.59	**33.80**
	3,281	35.56	**40.65**
ONCE	2,741	29.65	**34.71**
	3,862	31.90	**37.76**
	2,983	32.75	**38.64**
	4,638	32.76	**39.15**
	5,264	31.17	**37.66**

(2) Storage Consumption. The experimental results on storage consumption are presented in Tbl. 3.

Table 3: Storage consumption across different PC sequences.

Dataset	|Seq|	Storage Consumption (MB)				
		RS	Octree	LASzip	DUSTER-S2	DUSTER
Semantic-KITTI	4,541	8418.47	4310.34	2196.11	1846.20	**1812.92**
	4,661	8943.94	4760.32	2308.36	1862.41	**1806.58**
	4,071	7623.43	3958.04	2032.21	1744.41	**1710.28**
	4,982	9227.01	4672.43	2288.21	1896.18	**1844.95**
	3,281	5810.61	2934.62	1448.49	1272.90	**1255.98**
ONCE	2,741	2565.99	1218.32	1160.61	930.41	**888.48**
	3,862	3606.17	1662.27	1562.74	1235.06	**1170.66**
	2,983	2743.40	1274.09	1229.53	959.41	**905.82**
	4,638	4041.00	1860.62	1789.89	1384.31	**1300.28**
	5,264	4692.12	2176.64	2042.59	1612.82	**1515.07**

Both DUSTER-S2 and DUSTER outperform the baselines on SemanticKITTI and ONCE. Specifically, on the SemanticKITTI dataset, DUSTER achieves an average of 78.84% reduction in storage consumption (ranging from 77.56% to 80.00%) across five PC sequences, compared to the raw storage (RS) baseline. DUSTER also outperforms the best-performing baseline, LASzip, achieving an additional average reduction of 17.54% (ranging from 13.29% to 21.73%). DUSTER-S2 incurs higher storage consumption than DUSTER since it conducts 2-frame deduplication for the whole PC sequence. The results demonstrate that DUSTER provides stable and significant storage reduction across different sequences. Notably, we observe that storage consumption is not strictly proportional to the deduplication point percentage. This is due to two reasons: (1) additional feature data such as light intensity is also stored but not optimized by our deduplication method, and (2) LASzip compression performance is not linear w.r.t. the number of points, as larger batches of points tend to achieve better compression ratios. Since our method

[1] https://anonymous.4open.science/r/pc_storage_optimize-9E89

partitions PC frames into multiple PC patches, small patches may reduce the effectiveness of LASzip. This observation suggests that PC patches containing very few points should be removed when using LASzip as the compression backbone.

On the ONCE dataset, DUSTER shows consistent performance, outperforming the baselines similarly to the results on SemanticKITTI. DUSTER achieves an average storage reduction of 67.09% (ranging from 65.37% to 67.71%) across five PC sequences compared to RS, and achieves an additional average reduction of 25.60% (ranging from 23.44% to 27.35%) compared to LASzip. We also observe that both LASzip, DUSTER-S2, and DUSTER achieve lower storage reduction on ONCE than on SemanticKITTI, whereas the Octree method achieves higher storage reduction on ONCE. This is because the ONCE dataset has a sparser point cloud density compared to SemanticKITTI. LASzip performs better on dense point clouds due to higher compression efficiency, while Octree benefits from sparsity by requiring fewer levels in its tree structure. Furthermore, DUSTER achieves relatively better storage reduction over LASzip on the ONCE dataset than on SemanticKITTI, indicating that DUSTER effectively enhances LASzip compression performance under sparse point cloud conditions.

RQ2. Data Retrieval I/O Cost Evaluation. We evaluate the data retrieval performance for both random and sequential PC frame retrieval on different PC sequences (Seq 1 to Seq 5) from the SemanticKITTI dataset . In terms of loading latency for random retrieval, LASzip, DUSTER-S2, and DUSTER achieve efficient frame loading and reconstruction, each taking approximately 0.05 seconds per frame. In contrast, the Octree-based method incurs higher latency of around 10 seconds per frame due to the overhead of recursively reconstructing the tree structure.

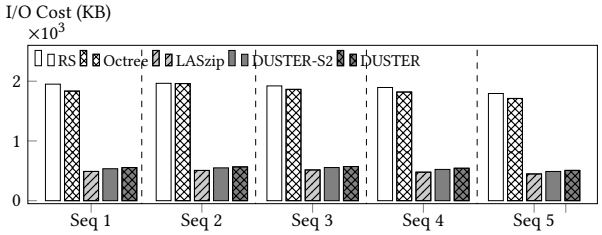


Figure 6: The I/O cost of random PC frame retrievals.

The experimental results on I/O cost for random PC frame retrieval are presented in Fig. 6. The LASzip, DUSTER-S2, and DUSTER methods achieve significant I/O cost reduction compared to the raw storage baseline. On average, LASzip requires 487.10 KB to load a single PC frame, while DUSTER-S2 (resp. DUSTER) requires 530.31 KB (resp. 548.54 KB), saving over 70% of I/O cost compared to RS. DUSTER incurs approximately 8% higher I/O cost than LASzip due to the need to load additional metadata for reconstructing the index structure, as it jointly manages neighboring PC frames (see Section 5.3). This overhead can be mitigated by caching metadata in memory. Although the Octree method reduces storage consumption relative to RS, it suffers from high I/O cost during random retrieval. This is because the method requires loading the entire Octree structure to reconstruct a PC frame. When only one frame is requested, the Octree still loads data associated with multiple frames, resulting in additional I/O overhead.

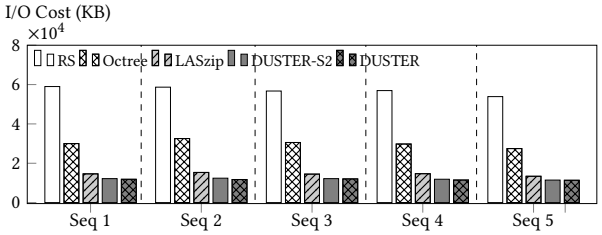


Figure 7: The I/O cost of sequential PC frame retrievals.

The experimental results for sequential PC frame retrieval are shown in Fig. 7. The results demonstrate that Octree, LASzip, DUSTER-S2, and DUSTER all achieve significant I/O cost reduction compared to the raw storage during sequential retrieval. Specifically, Octree, LASzip, DUSTER-S2, and DUSTER achieve 47.28%, 74.58%, 78.92%, and 79.49% I/O cost reduction, respectively. Unlike the random PC frame retrieval scenario, Octree performs well under sequential retrieval because the retrieval range often overlaps with the full Octree segments, allowing efficient loading of multiple frames together. Notably, DUSTER-S2 and DUSTER outperform LASzip in sequential retrieval, with DUSTER-S2 (resp. DUSTER) achieving an additional 20.58% (resp. 23.29%) I/O cost reduction over LASzip. This performance gain arises because DUSTER-S2 and DUSTER jointly store PC frames within segments, and under sequential retrieval, these frames are fully loaded (similar to Octree). This joint management becomes advantageous when the retrieval range aligns with the segment boundaries.

RQ3. Deep Model Accuracy Evaluation. We evaluate whether the reconstructed PC frames can maintain model prediction accuracy comparable to the original data. The accuracy metric is computed as follows: we use the bounding boxes predicted from the original PC frames as the ground truth, and measure the percentage of these ground-truth boxes that can be accurately predicted when using the reconstructed PC frames as input. The results of the deep model prediction accuracy are presented in Fig. 8.

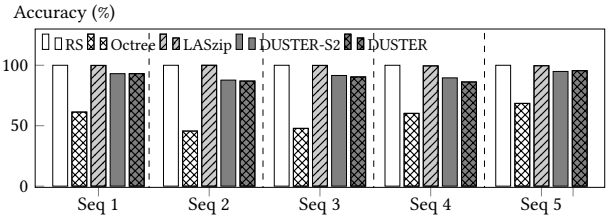
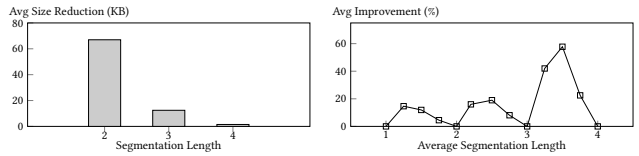


Figure 8: Deep model accuracy on reconstructed PC frames.

As shown in Fig. 8, the object detection accuracy using the PV-RCNN model on reconstructed PC frames reaches 56.67%, 99.77%, 91.36%, and 90.43% for Octree, LASzip, DUSTER-S2, and DUSTER, respectively. LASzip, DUSTER-S2, and DUSTER achieve high prediction accuracy, demonstrating that their reconstruction process does not significantly compromise model performance on downstream tasks. In contrast, the Octree-based method yields noticeably lower prediction accuracy. This performance gap is likely due to the Octree's difficulty in preserving fine-grained spatial resolution in regions where point density is high–an area where accurate object detection heavily relies on precise spatial details. As a result,

the reduced coordinate precision from the Octree reconstruction hinders the model's ability to accurately detect objects.

RQ4. Evaluation of Segmentation Method. We evaluate the effectiveness of our segmentation method and examine whether the proposed IP-MAB approach improves average deduplication performance under different computation budgets, i.e., with varying average segmentation lengths. Fig. 9a shows the average additional per-frame size reduction achieved when splitting all PC frames into longer segments compared to shorter ones (e.g., segment all frames into 2-frame segment instead of 1). Fig.9b presents the relative improvement of the IP-MAB method over the corresponding random segmentation baseline. Notably, when average segmentation length $x = 1, 2, 3, 4$, all frames are grouped into fixed-length segments of 1, 2, 3, or 4 frames, respectively. In these cases, there is no improvement of IP-MAB over the baseline, since the method degenerates into the fixed segmentation baselines DUSTER-S1 to S4 without adaptive allocation.



(a) Size reduction per frame. **(b) Performance improvement.**

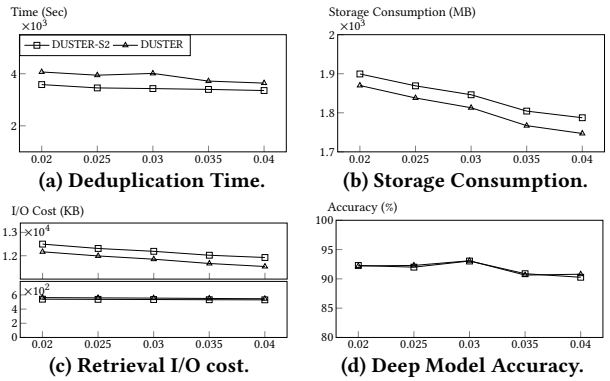
Figure 9: Performance of the segmentation method under varying average segmentation length.

As shown in Fig. 9a, splitting the PC sequence into 2-frame segments instead of 1-frame segments yields an average size reduction of 66.97 KB per frame. Increasing the segment length from 2 to 3 frames and from 3 to 4 frames provides additional reductions of 12.44 KB and 1.48 KB, respectively. As shown in Fig. 9b, With a time budget that allows 25% of the PC frames to be grouped into 2-frame segments (at $x = 1.25$), our IP-MAB method achieves a 14.64% improvement (i.e., 76.77 KB) in frame size reduction. When the budget allows 50% of the frames to be grouped into 3-frame segments (at $x = 2.5$) or 4-frame segments (at $x = 3.5$), the improvements reach up to 18.98% and 57.74%, respectively. These results demonstrate that IP-MAB consistently outperforms the random segmentation strategy, delivering stable performance gains. Notably, as the segment length increases and the marginal benefit of longer segments diminishes (e.g., with 4-frame segmentation), IP-MAB yields even greater improvements by adaptively prioritizing high-impact regions.

RQ5. Effect of Distance Threshold. We evaluate the performance by varying the distance threshold and the soft threshold ratio used in the deduplication process.

(1) Varying Distance Threshold. We evaluate the effect of the distance threshold by varying its value from 0.02 m to 0.04 m. We measure the changes in deduplication time cost, storage consumption, retrieval I/O cost (for both random and sequential retrieval), and deep model prediction accuracy as the distance threshold increases. The experimental results are presented in Fig. 10.

As reported in Fig. 10a, the deduplication time for both DUSTER-S2 and DUSTER decreases as the distance threshold grows. Specifically, the deduplication time of DUSTER slightly decreases from 4.0×10^3 s at a threshold of 0.02 m to 3.6×10^3 s at 0.04 m, while



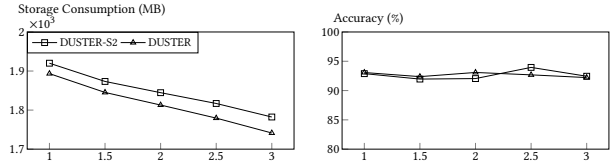
(a) Deduplication Time. **(b) Storage Consumption.**
(c) Retrieval I/O cost. **(d) Deep Model Accuracy.**

Figure 10: The effect of varying distance threshold from 0.02 m to 0.04 m. The lower (resp. upper) part of Fig. 10c reports the random (resp. sequential) retrieval I/O cost.

DUSTER-S2 decreases from 3.6×10^3 s to 3.35×10^3 s over the same range. Regarding storage consumption, shown in Fig. 10b, both DUSTER-S2 and DUSTER achieve lower storage usage as the distance threshold increases. DUSTER-S2 consumes more storage space than DUSTER since it groups all PC frames into 2-frame segments. In terms of I/O cost, as presented in Fig. 10c, the I/O cost for random retrieval (shown in the lower part of the figure) remains relatively stable and exhibits a slight decrease across varying distance thresholds. In contrast, the I/O cost for sequential retrieval decreases proportionally with the reduction in storage consumption as the threshold increases. The accuracy of deep model predictions with DUSTER (Fig. 10d), also affected by the distance threshold, slightly decreases from 93.0% to 90.3% as the threshold increases, indicating a minor trade-off between deduplication aggressiveness and model precision.

In summary, if the framework prioritizes high model prediction accuracy, a lower distance threshold is recommended. Conversely, for scenarios that emphasize higher sequential retrieval performance and reduced storage consumption, a higher distance threshold is preferable.

(2) Varying Soft Distance Threshold Ratio. We evaluate the effect of soft distance ratio by varying the number from 1.0 to 3.0. The results of storage consumption and accuracy are reported in Fig. 11.

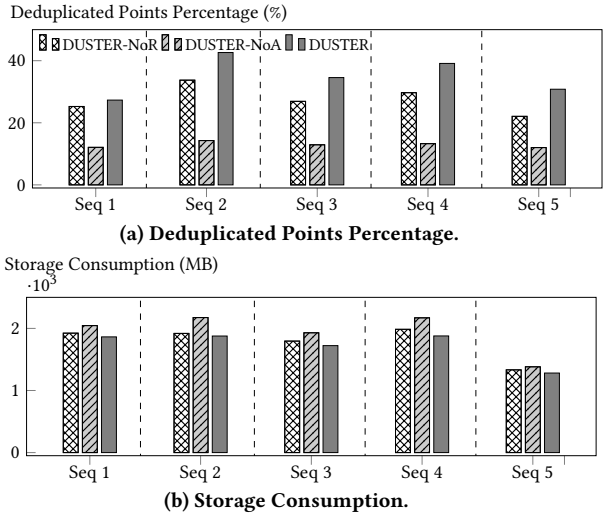


(a) Storage Consumption. **(b) Deep Model Accuracy.**

Figure 11: The effect of varying soft distance threshold ratio from 1 to 3 on storage consumption and accuracy.

As shown in Fig 11a, the trend of storage consumption changes tend to be similar as varying the distance threshold. However, the accuracy of deep model remains stable (from 93.1% to 92.2%)as the ratio increases, as shown in Fig. 11b. This demonstrate that the soft distance threshold ratio do enhance the deduplication performance while maintains the correctness of the reconstructed PC frames with the average chamfer distance unchanged.

RQ6. Redundancy Evaluation. We evaluate the effectiveness of the two-step deduplication process in the relative and absolute coordinate systems. Specifically, we compare two DUSTER variants: (1) DUSTER-NoR: It disables deduplication in the relative coordinate system and performs deduplication only in the absolute coordinate system. (2) DUSTER-NoA: It disables deduplication in the absolute coordinate system and performs deduplication only in the relative coordinate system. The deduplication point percentages and storage consumption results for DUSTER-NoR, DUSTER-NoA, and the full DUSTER framework are reported in Tbl. 12a and Tbl. 12b.



(a) Deduplicated Points Percentage.

(b) Storage Consumption.

Figure 12: Performance comparison on DUSTER variants.

According to the results reported in Fig. 12a, DUSTER-NoR achieves an average deduplication point percentage of 27.53% (ranging from 22.07% to 33.73%), while the full DUSTER framework achieves 34.88% on average (ranging from 27.31% to 42.63%). This means that DUSTER-NoR retains approximately 78.92% of the deduplication effectiveness of DUSTER. In contrast, DUSTER-NoA achieves a significantly lower average deduplication point percentage of 12.91% (ranging from 12.01% to 14.27%), which corresponds to 37.01% of DUSTER's performance. Regarding storage consumption (Fig. 12b), DUSTER-NoR also outperforms DUSTER-NoA, requiring an average of 1791.20 MB (ranging from 1332.41 MB to 1985.29 MB), which is approximately 4% more storage space compared to DUSTER for storing a PC sequence. These results demonstrate that combining deduplication in both the relative and absolute coordinate systems significantly improves deduplication performance. Notably, DUSTER-NoA shows more effective deduplication performance on the SemanticKITTI dataset, suggesting that the relative coordinate system plays a critical role under scenarios with higher spatial-temporal continuity.

7 Conclusion

In this paper, we explore joint deduplication across PC frame sequences and present DUSTER, a framework that efficiently reduces redundant PC points within a given computational budget. Extensive experiments demonstrate that DUSTER significantly reduces storage consumption and I/O cost for PC frame retrieval while preserving deep model prediction accuracy.

References

[1] Md Ahmed Al Muzaddid and William J Beksi. 2022. Variable rate compression for raw 3d point clouds. In *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 8748–8755.

[2] Michael Armbrust, Tathagata Das, Liwen Sun, Burak Yavuz, Shixiong Zhu, Mukul Murthy, Joseph Torres, Herman van Hovell, Adrian Ionescu, Alicja Łuszczak, et al. 2020. Delta lake: high-performance ACID table storage over cloud object stores. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3411–3424.

[3] Sayan Bandyapadhyay, Tanmay Inamdar, Shreyas Pai, and Kasturi Varadarajan. 2019. A constant approximation for colorful k-center. *arXiv preprint arXiv:1907.08906* (2019).

[4] Harry G Barrow and Jay M Tenenbaum. 1981. Interpreting line drawings as three-dimensional surfaces. *Artificial intelligence* 17, 1-3 (1981), 75–116.

[5] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jürgen Gall. 2019. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, 9296–9306. https://doi.org/10.1109/ICCV.2019.00939

[6] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975), 509–517.

[7] Wei Cao, Yingqiang Zhang, Xinjun Yang, Feifei Li, Sheng Wang, Qingda Hu, Xuntao Cheng, Zongzhi Chen, Zhenjun Liu, Jing Fang, Bo Wang, Yuhui Wang, Haiqing Sun, Ze Yang, Zhushi Cheng, Sen Chen, Jian Wu, Wei Hu, Jianwei Zhao, Yusong Gao, Songlu Cai, Yunyang Zhang, and Jiawang Tong. 2021. PolarDB Serverless: A Cloud Native Database for Disaggregated Data Centers. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. ACM, 2477–2489. https://doi.org/10.1145/3448016.3457560

[8] Jiafeng Chen, Lu Yu, and Wenyi Wang. 2022. Hilbert space filling curve based scan-order for point cloud attribute compression. *IEEE Transactions on Image Processing* 31 (2022), 4609–4621.

[9] Ron Chrisley. 2003. Embodied artificial intelligence. *Artificial intelligence* 149, 1 (2003), 131–150.

[10] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. *CoRR* abs/2401.08281 (2024). https://doi.org/10.48550/ARXIV.2401.08281 arXiv:2401.08281

[11] Lukas Drexler, Jan Eube, Kelin Luo, Dorian Reineccius, Heiko Röglin, Melanie Schmidt, and Julian Wargalla. 2024. Connected k-Center and k-Diameter Clustering. *Algorithmica* 86, 11 (2024), 3425–3464.

[12] 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.

[13] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. 2021. Deep Learning for 3D Point Clouds: A Survey. *IEEE Trans. Pattern Anal. Mach. Intell.* 43, 12 (2021), 4338–4364. https://doi.org/10.1109/TPAMI.2020.3005434

[14] Brandon Haynes, Maureen Daum, Dong He, Amrita Mazumdar, Magdalena Balazinska, Alvin Cheung, and Luis Ceze. 2021. Vss: A storage system for video analytics. In *Proceedings of the 2021 International Conference on Management of Data*. 685–696.

[15] Brandon Haynes, Maureen Daum, Amrita Mazumdar, Magdalena Balazinska, Alvin Cheung, and Luis Ceze. 2020. VisualWorldDB: A DBMS for the Visual World. In *10th Conference on Innovative Data Systems Research, CIDR 2020, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2020/papers/p12-haynes-cidr20.pdf

[16] Yun He, Xinlin Ren, Danhang Tang, Yinda Zhang, Xiangyang Xue, and Yanwei Fu. 2022. Density-preserving deep point cloud compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2333–2342.

[17] James Hilton. 2020. Autonomous vehicles generate 1.2TB of data per day–the equivalent of 500 HD movies or 200,000 songs. https://www.ai-online.com/2020/02/autonomous-vehicles-generate-1-2tb-of-data-per-day-the-equivalent-of-500-hd-movies-or-200000-songs/. Accessed: 2025-04-30.

[18] Jing Huang and Suya You. 2016. Point cloud labeling using 3D Convolutional Neural Network. In *23rd International Conference on Pattern Recognition, ICPR 2016, Cancún, Mexico, December 4-8, 2016*. IEEE, 2670–2675. https://doi.org/10.1109/ICPR.2016.7900038

[19] Jinwoo Hwang, Minsu Kim, Daeun Kim, Seungho Nam, Yoonsung Kim, Dohee Kim, Hardik Sharma, and Jongse Park. 2022. CoVA: Exploiting Compressed-Domain analysis to accelerate video analytics. In *Proceedings of the 2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, 707–722.

[20] Martin Isenburg. 2013. LASzip: lossless compression of LiDAR data. *Photogrammetric engineering and remote sensing* 79, 2 (2013), 209–217.

[21] Chris L Jackins and Steven L Tanimoto. 1980. Oct-trees and their use in representing three-dimensional objects. *Computer Graphics and Image Processing* 14, 3 (1980), 249–270.

[22] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. 1996. Reinforcement Learning: A Survey. *J. Artif. Intell. Res.* 4 (1996), 237–285. https://doi.org/10.1613/JAIR.301

[23] 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.

[24] Richard M. Karp. 1972. Reducibility Among Combinatorial Problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA (The IBM Research Symposia Series)*. Plenum Press, New York, 85–103. https://doi.org/10.1007/978-1-4684-2001-2_9

[25] Susana Ladra, Miguel R. Luaces, José R Paramá, and Fernando Silva-Coira. 2024. Compact and indexed representation for LiDAR point clouds. *Geo-spatial Information Science* 27, 4 (2024), 1035–1070.

[26] Jiangneng Li, Haitao Yuan, Gao Cong, Han Mao Kiah, and Shuhao Zhang. 2025. MAST: Towards Efficient Analytical Query Processing on Point Cloud Data. *Proc. ACM Manag. Data* 3, 1 (2025), 52:1–52:27. https://doi.org/10.1145/3709702

[27] Jiangneng Li, Haitao Yuan, Jie Wang, Ziting Wang, Han Mao Kiah, and Gao Cong. 2025. Demonstrating MAST: An Efficient System for Point Cloud Data Analytics. In *Companion of the 2025 International Conference on Management of Data (SIGMOD-Companion '25)* (Berlin, Germany). ACM, New York, NY, USA, 4. https://doi.org/10.1145/3722212.3725099

[28] Vanessa Lin, Yongming Ge, Maureen Daum, Alvin Cheung, Brandon Haynes, and Magdalena Balazinska. 2021. Demonstration of Apperception: A Database Management System for Geospatial Video Data. *Proc. VLDB Endow.* 14, 12 (2021), 2767–2770. https://doi.org/10.14778/3476311.3476340

[29] Hao Liu and Raymond Chi-Wing Wong. 2025. On efficient 3D object retrieval. *VLDB J.* 34, 1 (2025), 4. https://doi.org/10.1007/S00778-024-00884-7

[30] Yihao Lu and Hao Tang. 2025. Multimodal Data Storage and Retrieval for Embodied AI: A Survey. *CoRR* abs/2508.13901 (2025). https://doi.org/10.48550/ARXIV.2508.13901 arXiv:2508.13901

[31] Jiageng Mao, Minzhe Niu, Chenhan Jiang, Hanxue Liang, Jingheng Chen, Xiaodan Liang, Yamin Li, Chaoqiang Ye, Wei Zhang, Zhenguo Li, Jie Yu, Chunjing Xu, and Hang Xu. 2021. One Million Scenes for Autonomous Driving: ONCE Dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

[32] n.d. 2023. PostgreSQL pgPointcloud. https://pgpointcloud.github.io/pointcloud/. [Online; accessed 27-Jan-2023].

[33] Gao Peng, Bo Pang, and Cewu Lu. 2021. Efficient 3D Video Engine Using Frame Redundancy. In *IEEE Winter Conference on Applications of Computer Vision, WACV 2021, Waikoloa, HI, USA, January 3-8, 2021*. IEEE, 3791–3801. https://doi.org/10.1109/WACV48630.2021.00384

[34] PostgreSQL Global Development Group. [n. d.]. PostgreSQL. https://www.postgresql.org/. Accessed: 2025-09-25.

[35] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2017. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 77–85. https://doi.org/10.1109/CVPR.2017.16

[36] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. 2017. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. (2017), 5099–5108. https://proceedings.neurips.cc/paper/2017/hash/d8bf84be3800d12f74d8b05e9b89836f-Abstract.html

[37] 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.

[38] Radu Bogdan Rusu and Steve Cousins. 2011. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011*. IEEE. https://doi.org/10.1109/ICRA.2011.5980567

[39] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. 2020. PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, 10526–10535. https://doi.org/10.1109/CVPR42600.2020.01054

[40] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. 2019. PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 770–779. https://doi.org/10.1109/CVPR.2019.00086

[41] Kaleem Siddiqi, Juan Zhang, Diego Macrini, Ali Shokoufandeh, Sylvain Bouix, and Sven J. Dickinson. 2008. Retrieving articulated 3-D models using medial surfaces. *Mach. Vis. Appl.* 19, 4 (2008), 261–275. https://doi.org/10.1007/S00138-007-0097-8

[42] Doug Stewart. 1965. A platform with six degrees of freedom. *Proceedings of the institution of mechanical engineers* 180, 1 (1965), 371–386.

[43] OpenPCDet Development Team. 2020. OpenPCDet: An Open-source Toolbox for 3D Object Detection from Point Clouds. https://github.com/open-mmlab/OpenPCDet.

[44] Matt P Wand and M Chris Jones. 1994. *Kernel smoothing*. CRC press.

[45] Tai Wang, Xiaohan Mao, Chenming Zhu, Runsen Xu, Ruiyuan Lyu, Peisen Li, Xiao Chen, Wenwei Zhang, Kai Chen, Tianfan Xue, et al. 2024. Embodiedscan:

A holistic multi-modal 3d perception suite towards embodied ai. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 19757–19767.

[46] Laurence A. Wolsey. 2020. *Integer Programming*. John Wiley & Sons.

[47] Wen Xia, Hong Jiang, Dan Feng, Fred Douglis, Philip Shilane, Yu Hua, Min Fu, Yucheng Zhang, and Yukun Zhou. 2016. A Comprehensive Study of the Past, Present, and Future of Data Deduplication. *Proc. IEEE* 104, 9 (2016), 1681–1710. https://doi.org/10.1109/JPROC.2016.2571298

[48] Jiong Xie, Zhen Chen, Jianwei Liu, Fang Wang, Feifei Li, Zhida Chen, Yinpei Liu, Songlu Cai, Zhenhua Fan, Fei Xiao, and Yue Chen. 2022. Ganos: A Multidimensional, Dynamic, and Scene-Oriented Cloud-Native Spatial Database Engine. *Proc. VLDB Endow.* 15, 12 (2022), 3483–3495. https://doi.org/10.14778/3554821.3554838

[49] Yan Yan, Yuxing Mao, and Bo Li. 2018. SECOND: Sparsely Embedded Convolutional Detection. *Sensors* 18, 10 (2018), 3337. https://doi.org/10.3390/S18103337