International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Spatial Locality as the Governing Design Constraint for Petabyte-Scale Point Cloud Platforms

Prateek Jindal

Abstract

Point cloud platforms built for autonomous vehicles, robotics, and digital twin systems tend to inherit their architecture from general-purpose big data infrastructure, with spatial partitioning bolted on as an implementation detail rather than treated as a governing constraint. This article takes the opposite position: spatial locality — the principle that data representing nearby physical regions should sit close together in storage and retrieval paths — belongs at the center of point cloud platform design, from ingestion through retrieval, not layered on as an afterthought. The argument traces how point cloud infrastructure draws on two separate lineages, spatial indexing structures and distributed storage systems, without being fully served by either on its own. It examines spatial indexing as the foundational design problem, distributed storage architecture adapted for spatial partitioning, metadata as the coordination layer governing usability at scale, and retrieval and visualization as the point where these design choices pay off or fail. The discussion extends to autonomous transportation, robotics, and digital twins, framed as consumers of a well-designed spatial platform rather than the source of its design requirements. Platforms treating spatial locality as secondary accumulate storage capacity while retrieval performance degrades as datasets grow — the reverse of what these workloads need.

Keywords: *Distributed Storage Systems, LiDAR Data Infrastructure, Point Cloud Data Management, Spatial Indexing, Spatial Partitioning*

1. Introduction

Three-dimensional sensing has moved out of research labs and into everyday infrastructure. LiDAR units on autonomous vehicles, robotic platforms, and mapping systems generate spatial observations continuously, and datasets that used to be research-scale curiosities now run into petabytes in production [12], [13], [14]. Much of the published attention on this growth focuses on what happens once point cloud data reaches a machine learning pipeline: how models consume it, which architectures process it, what accuracy comes out the other end. A prior question gets less attention, and this article argues it deserves more — how should the data be organized in the first place so that any of those downstream uses is even possible at the volumes involved?

Most production point cloud platforms borrow infrastructure patterns wholesale from general-purpose big data systems: object storage for durability, distributed processing frameworks for transformation, partitioning schemes built around arbitrary keys or time ranges rather than spatial coordinates. These patterns do their job well in the contexts they were designed for. They fit point cloud data poorly, at least unmodified, because point cloud consumption is overwhelmingly spatial in character. A robot querying its immediate surroundings, an analyst inspecting a single city block, an autonomous vehicle validating one specific intersection — all three need data organized by where it physically is, not by when it arrived or which arbitrary shard happened to catch it.

Spatial locality is the idea at the center of this article: data representing nearby physical regions should be kept physically close together in storage and retrieval paths, and that principle should govern point cloud platform design from the ingestion layer through metadata management to retrieval and visualization. Generic distributed storage and processing patterns remain useful building blocks. They just need adapting first. A platform that skips this adaptation accumulates capacity without gaining the retrieval performance that actually determines whether the system is usable once it is large.

Independent Researcher, USA
ORCID ID: 0009-0002-8459-1233

Four sections carry this argument forward before the discussion turns outward. Section 2 lays out why point cloud workloads violate assumptions baked into generic data infrastructure. Section 3 argues that spatial indexing, not storage format, is the foundational design problem. Section 4 traces how distributed storage architecture has to be adapted, not just applied, once partition boundaries need to track spatial locality. Section 5 turns to metadata as the coordination layer that increasingly determines whether a platform stays usable as it grows. Section 6 examines retrieval and visualization at scale as the practical test of everything that came before, Section 7 extends the argument to industry applications, and Section 8 closes.

2. Why Point Cloud Workloads Break Generic Data Infrastructure Assumptions

Object stores, distributed file systems, and processing frameworks — the standard toolkit of cloud-native architecture — were built around workloads with access patterns that look nothing like how point cloud data actually gets used. Four characteristics of point cloud workloads make that mismatch concrete.

Volume is the most visible of the four, though not the one doing the most architectural damage on its own. A single autonomous vehicle can produce terabytes of sensor data in a day, and a fleet operating at any real scale reaches petabytes within a short collection window [12], [13]. Systematic reviews of point cloud data management approaches document the same scaling pressure across a range of deployment contexts, not just autonomous driving specifically [15]. The Waymo Open Dataset and nuScenes both document multimodal sensor suites producing continuous streams at rates that make a naive store-and-forget approach impractical past a certain fleet size [12], [13]. Volume by itself, though, is a problem generic big data infrastructure already handles reasonably well. The harder problem starts once you ask what happens next.

Three-dimensional structure is where generic infrastructure assumptions start to come apart. Most enterprise data systems are built around structured records — rows, documents, key-value pairs — where a record's identity has nothing to do with its physical proximity to other records. Point clouds represent geometric information spread across three-dimensional space, and two points can be logically unrelated (captured by different sensors on different days) while still sitting right next to each other spatially. That spatial adjacency is frequently the property a query actually cares about. A storage system indexed by ingestion time or an arbitrary shard key has no efficient way to answer "give me everything near this coordinate" — it can only answer that question by scanning far more data than the query logically needs.

Access patterns make the problem worse. Point cloud data is rarely consumed sequentially. Users and downstream systems navigate through captured environments, zooming into regions of interest and pulling bounded spatial subsets rather than the whole dataset. That interactive, region-bounded pattern is fundamentally different from the batch, full-scan pattern that frameworks like MapReduce were originally built to optimize [6], and it calls for an index structure that can answer spatial range queries efficiently — not merely a partitioning scheme that can parallelize a full scan faster.

Multi-modal integration adds a further layer. Point cloud platforms typically combine LiDAR returns, camera imagery, GPS or inertial positioning, and sensor metadata into one logical dataset [13]. KITTI and its successors made this kind of multi-sensor fusion a standard expectation for autonomous-driving datasets rather than an edge case [4], [13], [14]. Keeping track of relationships among these different data types, while still preserving the spatial coherence needed to retrieve "everything relevant to this location, across every sensor" in a single operation, is a coordination problem generic infrastructure was never built to solve directly.

Put these four together and it becomes clear that point cloud infrastructure cannot simply inherit a generic big data architecture and expect acceptable retrieval performance out of it. The next section argues that fixing this starts with treating spatial indexing, not storage format, as the platform's foundational design problem.

3. Spatial Indexing as the Foundational Design Problem

A common instinct in point cloud platform design is to treat storage format — object storage versus block storage, columnar versus row-oriented — as the primary architectural decision, with spatial organization bolted on afterward through auxiliary indexes or metadata tags. That ordering gets the problem backward. Storage format is a relatively interchangeable implementation detail. Spatial indexing is the structural decision that determines

whether the platform can actually answer the access patterns described in Section 2 without moving far more data than necessary.

This lineage predates point cloud computing by decades. Bentley's k-d tree, built originally as a general-purpose structure for associative searching in multidimensional data, established the core idea that spatial partitioning could be represented as a recursive binary decomposition of space — enough to enable efficient nearest-neighbor and range queries without exhaustive search [1]. Octree encoding extended a related idea into a hierarchical, cube-based decomposition suited to three-dimensional volumetric data, and it became a widely-adopted structural pattern among the point cloud processing tools that followed [2]. The Point Cloud Library's decision to build octree-based spatial indexing in as a core capability, rather than bolt it on as an optional extension, shows how central this lineage is to practical point cloud tooling: PCL treats spatial partitioning as infrastructure, not as an analysis step [3].

What generic distributed storage systems inherited instead was a different partitioning philosophy entirely — hash-based or range-based partitioning, designed to spread load evenly across nodes without regard to any semantic relationship between neighboring keys. That is a sensible default for structured or transactional workloads, where neighboring keys often have no meaningful relationship to each other at all. It is a poor default for point cloud data, where spatial adjacency is frequently the exact relationship a query is trying to exploit. Spatially-aware partitioning substantially reduces the volume of data scanned per query relative to arbitrary key-based schemes, which is precisely the property that determines whether retrieval remains tractable as a dataset grows.

Table 1 lays out the comparison between the three spatial indexing approaches most commonly used in point cloud platforms.

Table 1: Spatial Indexing Structures Compared

Structure	Strengths	Weaknesses
K-d tree	Efficient nearest-neighbor and range queries; well-suited to point data	Rebalancing cost under frequent updates; less natural fit for volumetric data
Octree	Natural fit for 3D volumetric decomposition; hierarchical level-of-detail support	Can be memory-intensive at fine resolutions; imbalance under non-uniform density
Hierarchical grid	Simple to implement; predictable partition boundaries	Less adaptive to variable point density than octrees or k-d trees

The practical takeaway is that a point cloud platform's partitioning scheme should derive from a spatial index — an octree, a k-d tree variant, or a hierarchical grid — rather than from an arbitrary key range that happens to be convenient for whatever storage system sits underneath. Section 4 looks at what that means for how distributed storage architecture itself needs to change.

4. Distributed Storage Architecture: Object Storage Meets Spatial Partitioning

Modern distributed storage architecture has its own well-documented lineage. The Google File System set the pattern for chunk-based, replicated storage designed for large files and high-throughput sequential access [5]. Bigtable extended that into a distributed storage system for structured, sparse data organized by row keys, built around an assumption that applications would exercise control over key structure to influence how data lands on disk [9]. Spanner pushed the idea further, into a globally-distributed, strongly-consistent database that still relies on careful key design to manage how data gets sharded across regions [10]. MapReduce and its successor, the Resilient Distributed Dataset abstraction behind Apache Spark, built the processing-side counterpart: split data across a cluster, run transformations in parallel, and survive node failure through recomputation rather than relying solely on replication [6], [7].

None of these systems were built with spatial coordinates as the organizing key. Their underlying mechanics turn out to be adaptable, though, and that adaptability is exactly what a well-designed point

cloud platform exploits. Bigtable's row-key design already assumes applications will engineer key structure to influence locality — the same mechanism can be repurposed to encode a spatial index value, a Morton code or Hilbert curve position derived from an octree cell, for instance, as the row key itself, so spatially adjacent points end up in adjacent storage locations instead of arbitrary ones. Spark's RDD abstraction does not require partitions to correspond to time ranges or hash buckets either; a point cloud platform can partition an RDD by spatial index cell instead, keeping the same fault-tolerance and parallelism guarantees while aligning partition boundaries with the access pattern that actually matters.

Recent systems work confirms this adaptation is happening in practice, not just as a theoretical proposal. Work on point cloud data management within a lakehouse architecture shows how spatially-aware metadata and partitioning can be layered onto modern lakehouse storage formats, combining the operational maturity of established distributed storage patterns with organization that actually respects spatial locality [16]. Earlier work on point cloud management within a big data paradigm reached a broadly similar conclusion using an explicitly spatial partitioning scheme layered on distributed processing infrastructure — suggesting this is less a novel insight than a recurring requirement that different teams keep arriving at independently [19].

Table 2: Distributed Storage Precedents Mapped to Point Cloud Infrastructure Concerns

System	**Core Contribution**	**Point Cloud Infrastructure Concern Addressed**
Google File System	Chunk-based, replicated storage for large files	Durability and high-throughput sequential ingestion
Bigtable	Row-key-controlled data locality	Repurposable for spatial-index-derived row keys
Spanner	Globally-distributed, strongly-consistent database	Consistency at scale across regions/partitions
MapReduce / Spark (RDD)	Parallel processing with fault tolerance via recomputation	Adaptable to spatial-index-cell-based partitioning

Figure 1: Point Cloud Platform Architecture — Ingestion Through Retrieval, Organized Around Spatial Partitioning

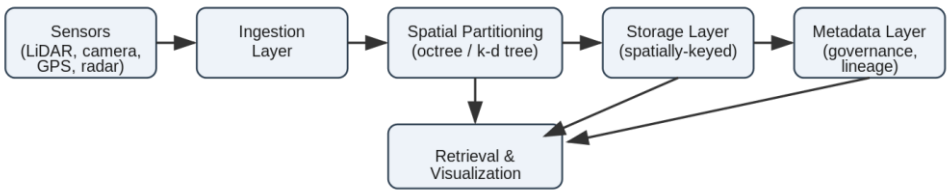

Figure 1: Point Cloud Platform Architecture — Ingestion Through Retrieval, Organized Around Spatial Partitioning

None of this argues that point cloud platforms need bespoke distributed systems built from scratch. The point is narrower: the storage and processing layers borrowed from general-purpose big data infrastructure need deliberate adaptation at the key-design and partitioning level, and that adaptation only becomes possible once spatial indexing, as argued in Section 3, is already established as the structure the storage layer is built around.

A concrete illustration helps here. Consider a metropolitan mapping fleet collecting continuous LiDAR sweeps across a city over several months. Under a naive time-partitioned scheme, a query for "everything captured within this three-block radius"

would need to scan every partition spanning the entire collection period, since nothing about the storage layout reflects geography. Under a spatially-partitioned scheme keyed to octree cells, the same query touches only the partitions covering that specific radius, regardless of when each sweep happened to occur. The difference is not a marginal optimization — it is the difference between a query that scales with dataset size and one that scales with query scope, which is the actual property a usable platform needs as the fleet's collection history grows indefinitely.

5. Metadata as the Governing Coordination Layer

A pattern recurs across point cloud platform deployments as they scale: metadata complexity grows faster than raw data volume, a pattern documented in lakehouse-based point cloud management work specifically [16]. That is not the intuitive expectation going in — most people assume the dominant scaling challenge will be the sheer size of the point cloud data itself — but it reflects how much coordination work metadata ends up doing in a mature platform. Metadata tracks geographic boundaries, sensor identifiers, collection timestamps, calibration parameters, and dataset lineage, and every one of those dimensions multiplies as a platform ingests data from more sensors, more collection campaigns, and more downstream consumers with different retrieval needs.

Lakehouse-oriented approaches to point cloud data management treat this head-on, layering governance, lineage, and data quality metadata directly onto the spatially-partitioned storage layer instead of managing metadata as a separate system bolted on after the fact [16]. This integration matters because a metadata layer that lives architecturally apart from the spatial index risks drifting out of sync with it — a common failure mode where the metadata claims one spatial extent for a dataset while the actual partitioning reflects a different, stale boundary.

Point cloud compression standardization work is metadata-adjacent in a way that is easy to miss. The ongoing standardization of video-based and geometry-based point cloud compression addresses how point cloud data itself gets encoded, but the metadata that goes along with it — which encoding scheme applies, at what fidelity, over what spatial extent — becomes part of the same coordination problem that governs retrieval [8]. A platform that cannot answer "what compression scheme applies to this spatial region" as reliably as it answers "what points exist in this spatial region" has not solved metadata management. It has postponed part of it.

The practical implication is that metadata deserves treatment as a first-class coordination layer governed by the same spatial-locality principle as the storage layer, rather than as a secondary bookkeeping system that references the storage layer from the outside. Section 6 turns to retrieval and visualization, where the payoff — or failure — of this design choice becomes directly visible.

6. Retrieval and Visualization at Scale

Retrieval and visualization performance is where a spatial-locality-aware design either pays off or exposes itself as inadequate, because these are the operations users and downstream systems actually experience directly, not abstractions buried several layers down. Processing an entire point cloud dataset for a single query stops being practical once volume reaches production scale; systems instead need to retrieve only the spatial subset a given task actually requires.

Out-of-core, multiresolution retrieval strategies tackle this directly by pairing a spatial index with an adaptive level-of-detail scheme, so interactive visualization can render an approximation of a large spatial region quickly and sharpen it progressively as a user zooms or navigates, rather than loading full-resolution data for the entire dataset up front [17]. This approach depends entirely on the spatial index established earlier in the pipeline. Without it, there is no principled way to decide which subset of the data corresponds to the region a user is actually looking at.

Distributed visualization work extends a similar principle to civil-engineering-scale point cloud datasets, using a distributed computing approach to render big point cloud datasets that exceed what a single machine could hold or process, again relying on the underlying data being partitioned so that spatial subsetting stays tractable across the distributed system rather than requiring a full-dataset scan [18].

Figure 2: Retrieval Flow — From Client Query Through Spatial Index to Progressive Rendering

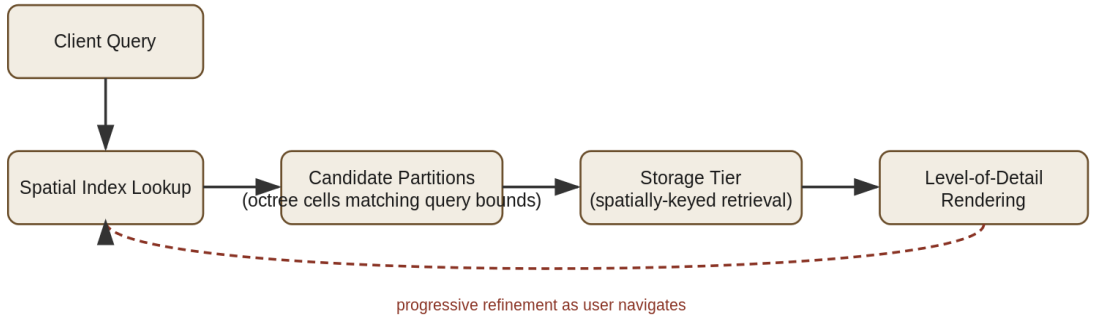

Figure 2: Retrieval Flow — From Client Query Through Spatial Index to Progressive Rendering

Both lines of work point to the same underlying fact: retrieval and visualization performance is not primarily a rendering-engine problem or a network-bandwidth problem, though both matter at the margins. It is fundamentally a consequence of whether the storage and metadata layers beneath the visualization system were organized around spatial locality to begin with. Get Sections 3 through 5 right, and Section 6 becomes tractable. Treat spatial organization as an afterthought, and no amount of rendering-engine tuning will compensate for a storage layer that cannot answer spatial range queries efficiently.

7. Industry Applications and Implications for AI/ML-Consuming Workloads

The architectural argument built across Sections 2 through 6 has direct consequences for the industries that depend on point cloud infrastructure, though the direction of that dependency is worth stating precisely. Autonomous vehicle fleets, industrial robotics, and digital twin systems consume point cloud platforms — they do not define the platform's core design requirements. Conflating "what autonomous vehicles need" with "how point cloud platforms should be designed" is part of what leads platforms to get architected around machine learning consumption patterns instead of the spatial retrieval patterns that actually determine whether the platform works at all.

Autonomous transportation systems rely on point cloud infrastructure to store, retrieve, and continuously refine the perception datasets that train and validate driving models, and the growth of benchmark datasets in this space — from KITTI's original scale up through the substantially larger Waymo Open Dataset, nuScenes, and Argoverse 2 collections — tracks the same volume pressure described in Section 2 [4], [12], [13], [14]. The growing consumption of point cloud data by machine learning training pipelines is genuinely a major driver of the volume and access-pattern pressures this article addresses, even though the argument here concerns the infrastructure underneath those pipelines rather than the pipelines themselves.

Robotics platforms operating in dynamic environments depend on spatial retrieval fast enough to support real-time navigation decisions, which puts the retrieval-latency concerns from Section 6 directly in the critical path of the robot's operation rather than treating them as a background analytics concern.

Digital twin systems, which maintain continuously updated virtual representations of physical environments, depend on efficient incremental updates to spatially-organized data — a requirement that maps directly onto the metadata-coordination concerns from Section 5, since a digital twin is only useful if its spatial and temporal metadata stays synchronized with the underlying point cloud updates as they arrive.

These applications share one dependency in common: none of them are well served by a platform that scales in raw storage capacity while its spatial retrieval performance degrades. The architectural principles developed here function as a precondition for these applications to work at production scale, not as an optimization bolted onto systems that already function.

8. Conclusion

Point cloud platforms are frequently designed as extensions of general-purpose big data infrastructure, with spatial organization treated as a secondary concern addressed through auxiliary indexes layered onto storage systems built around

arbitrary partitioning. This article has argued that ordering is backward: spatial locality should govern platform design from ingestion through retrieval, with generic distributed storage and processing systems adapted to serve that constraint rather than applied unmodified and patched afterward. The lineage running from k-d trees and octrees through modern lakehouse-based point cloud management systems suggests this is not a novel proposal so much as a recurring requirement that different implementations keep arriving at independently. Platforms that get this ordering right keep retrieval and visualization tractable as datasets grow. Platforms that treat spatial organization as an afterthought accumulate capacity without gaining the performance that actually determines whether the system stays usable. As autonomous systems, robotics, and digital twin applications keep scaling their data requirements, the platforms underneath them will be judged less by how much they can store and more by how efficiently they answer the spatial questions their consumers actually ask.

Declarations

CRediT Author Contributions: Prateek Jindal — Conceptualization, Methodology, Investigation, Writing – Original Draft, Writing – Review & Editing, Visualization.

Conflicts of Interest: The author declares no conflict of interest.

Funding: No funding was received for this research.

Statement of Human and Animal Rights: This study did not involve human participants, animal subjects, or identifiable personal data requiring ethics board review.

Informed Consent: Not applicable.

Generative AI Use Disclosure: The author used Claude (Anthropic, claude-sonnet model) to assist with manuscript drafting and language refinement, including a humanization and AI-writing-pattern audit pass. The author takes full responsibility for the accuracy, originality, and integrity of all content presented in this manuscript, and confirms that all data, analysis, and conclusions reflect independent work and verified sources.

References

[1] Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. Communications of the ACM, 18(9), 509–517. https://doi.org/10.1145/361002.361007

[2] Meagher, D. (1982). Geometric modeling using octree encoding. Computer Graphics and Image Processing, 19(2), 129–147.

[3] Rusu, R. B., & Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA), 1–4.

[4] Geiger, A., Lenz, P., & Urtasun, R. (2012). Are we ready for autonomous driving? The KITTI vision benchmark suite. Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 3354–3361.

[5] Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The Google file system. Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP), 29–43. https://doi.org/10.1145/945445.945450

[6] Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified data processing on large clusters. Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 137–150.

[7] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., & Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 15–28.

[8] Graziosi, D., Nakagami, O., Kuma, S., Zaghetto, A., Suzuki, T., & Tabatabai, A. (2020). An overview of ongoing point cloud compression standardization activities: Video-based (V-PCC) and geometry-based (G-PCC). APSIPA Transactions on Signal and Information Processing, 9, e13. https://doi.org/10.1017/ATSIP.2020.12

[9] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., & Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems, 26(2), Article 4. https://doi.org/10.1145/1365815.1365816

[10] Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J. J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., Hsieh, W., Kanthak, S., Kogan, E., Li, H., Lloyd, A., Melnik, S., Mwaura, D., Nagle, D., Quinlan, S.,

Rao, R., Rolig, L., Saito, Y., Szymaniak, M., Taylor, C., Wang, R., & Woodford, D. (2012). Spanner: Google's globally-distributed database. Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 251–264.

[11] Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., Ghodsi, A., Gonzalez, J., Shenker, S., & Stoica, I. (2016). Apache Spark: A unified engine for big data processing. Communications of the ACM, 59(11), 56–65. https://doi.org/10.1145/2934664

[12] Sun, P., Kretzschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., Guo, J., Zhou, Y., Chai, Y., Caine, B., Vasudevan, V., Han, W., Ngiam, J., Zhao, H., Timofeev, A., Ettinger, S., Krivokon, M., Gao, A., Joshi, A., Zhang, Y., Shlens, J., Chen, Z., & Anguelov, D. (2020). Scalability in perception for autonomous driving: Waymo Open Dataset. Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2446–2454.

[13] Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., & Beijbom, O. (2020). nuScenes: A multimodal dataset for autonomous driving. Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 11621–11631.

[14] Wilson, B., Qi, W., Agarwal, T., Lambert, J., Singh, J., Khandelwal, S., Pan, B., Kumar, R., Hartnett, A., Pontes, J. K., Ramanan, D., Carr, P., & Hays, J. (2023). Argoverse 2: Next generation datasets for self-driving perception and forecasting. Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track.

[15] Lokugam Hewage, C. N., Laefer, D. F., Vo, A.-V., Le-Khac, N.-A., & Bertolotto, M. (2022). Scalability and performance of LiDAR point cloud data management systems: A state-of-the-art review. Remote Sensing, 14(20), 5277. https://doi.org/10.3390/rs14205277

[16] Teuscher, B., & Werner, M. (2025). Point cloud data management for analytics in a lakehouse. AGILE GIScience Series, 6, Article 47.

[17] Teijeiro, D., Amor, M., Doallo, R., & Deibe, D. (2023). Interactive visualization of large point clouds using an autotuning multiresolution out-of-core strategy. The Computer Journal, 66(7), 1802–1816. https://doi.org/10.1093/comjnl/bxac179

[18] Nguyen, M. H., Yoon, S., Ju, S., Park, S., & Heo, J. (2022). B-EagleV: Visualization of big point cloud datasets in civil engineering using a distributed computing solution. Journal of Computing in Civil Engineering, 36(3), Article 04022005. https://doi.org/10.1061/(ASCE)CP.1943-5487.0001021

[19] Pajić, V., Govedarica, M., & Amović, M. (2018). Model of point cloud data management system in big data paradigm. ISPRS International Journal of Geo-Information, 7(7), 265. https://doi.org/10.3390/ijgi7070265