Vol. 2, No. 4, pp.: 49-64, 2024

SpatioTemporalDBKit: Database Extensions for Moving-Object and Event-Based Analytics

Marek Kowalski[1], Anna Wisniewska[2], Piotr Nowak[1,*]

[1] Department of Computer Science, Silesian University of Technology, Gliwice 44-100, Poland

[2] Department of Geodesy and Cartography, Wroclaw University of Science and Technology, Wroclaw 50-370, Poland

* *piotr.nowak@polsl.pl*

Article Information	
Received	19 July 2024
Accepted	26 November 2024
DOI	https://doi.org/10.63646/datamind.2024.020405

Abstract

Spatial and temporal data streams generated by GPS-equipped vehicles, IoT sensor networks, urban mobility systems, and satellite tracking platforms collectively represent one of the fastest-growing categories of operational data in modern computing. Yet the relational database ecosystem still lacks a consolidated, schema-documented, and publicly available toolkit that transforms raw moving-object streams into analytically ready knowledge structures. This paper presents SpatioTemporalDBKit (STDBKit), a modular database extension library built on top of PostgreSQL and PostGIS that provides trajectory storage, spatiotemporal indexing, event detection, and window-based analytical functions for moving-object and event-based analytics. STDBKit introduces six normalized relational tables with complete field dictionaries and foreign-key integrity, a configurable five-stage ingestion pipeline covering coordinate-reference-system normalization, noise filtering, trajectory segmentation, Douglas-Peucker compression, and R*-tree or SP-GiST spatial indexing. An extension function layer delivers over 40 PL/pgSQL and Python user-defined functions for trajectory interpolation, spatial join, event-sequence detection, and sliding-window aggregation. Three benchmark experiments are reported: query latency scaling across dataset sizes from 0.5 M to 50 M GPS points; index size and build-time comparison against PostGIS and MobilityDB; and trajectory compression accuracy measured by Synchronized Euclidean Distance. On a 50 M-point benchmark corpus, STDBKit SP-GiST indexing reduces index size by 51% and build time by 47% relative to a vanilla PostGIS R-tree configuration, while achieving a median range-query latency of 38 ms. All source code, schema scripts, seed data, and Docker images are released under Apache 2.0.

Keywords: ***Spatiotemporal database; moving objects; trajectory analytics; spatial indexing; event detection; PostGIS; database extensions***

1. Introduction

The proliferation of location-aware devices has fundamentally altered the relationship between real-world movement and its digital representation. GPS receivers embedded in smartphones, freight vehicles, marine vessels, drones, and wildlife tracking collars now produce position fixes at sub-second intervals, generating data volumes that routinely exceed hundreds of gigabytes per day for a single urban mobility operator (Zheng, 2015; Andrienko and Andrienko, 2011). The scientific and operational challenges that follow from this abundance are well documented: how should raw positional streams be stored efficiently, how should duplicate and noisy observations be suppressed without losing behavioural signal, how should analytically meaningful units such as trips, stops, and manoeuvres be extracted automatically, and how should the resulting objects be queried at low latency across millions of records (Renso et al., 2013; Pelekis and Theodoridis, 2014)?

Relational database management systems (RDBMS) remain the dominant platform for operational data management across government, industry, and research institutions. PostgreSQL, in particular, has developed into a mature extensible platform whose PostGIS extension provides a rich set of geometry and geography types as well as standard spatial query functions (Guttman, 1984; Beckmann et al., 1990). However, PostGIS was designed primarily for static geometry rather than for objects that move through space over time. Queries that ask where an object was at a specific time, which objects shared a spatial region during a given interval, or whether an object crossed a geofence boundary require temporal predicates that neither standard SQL nor PostGIS natively supports without substantial user-side boilerplate (Erwig et al., 1999; Gueting and Schneider, 2005).

MobilityDB, a recent open-source extension to PostgreSQL, represents the most significant effort to close this gap by introducing native temporal types and temporal geometric predicates (Brinkhoff, 2002; Trajcevski et al., 2004). While MobilityDB provides an expressive type system, it imposes a non-trivial installation dependency chain and does not include the full analytical function library, data ingestion pipeline, noise filtering, or compression utilities that a practitioner needs to go from raw GPS files to production-ready analytics. A practitioner who adopts MobilityDB still faces the challenge of building ingestion adapters, quality-control routines, event detection logic, and result caching independently (Spaccapietra et al., 2008; Pelekis et al., 2007).

This paper presents SpatioTemporalDBKit (STDBKit), a modular extension library that addresses this gap at the architectural level rather than the type level. STDBKit is implemented as a set of PL/pgSQL schema scripts, Python user-defined functions (UDFs), and a companion CLI tool that together convert raw GPS or IoT streams into a relational structure suitable for both classical SQL analytics and modern machine-learning feature extraction. The design philosophy of STDBKit prioritizes three properties: (1) portability, meaning that the extensions run on any PostgreSQL 14+ installation with PostGIS 3.x without requiring custom compiled extensions; (2) reproducibility, meaning that every pipeline step is parameterized and logged in metadata tables so that experiments can be re-executed deterministically; and (3) openness, meaning that all code, schema definitions, seed datasets, and container images are publicly released under Apache 2.0.

The remainder of this paper proceeds as follows. Section 2 surveys existing spatiotemporal database gaps and the use cases that motivate STDBKit. Section 3 describes the data sources and schema design. Section 4 details the construction pipeline and extension function layer. Section 5 reports three benchmark experiments. Section 6 addresses reproducibility and open access. Section 7 discusses limitations, and Section 8 concludes.

2. Database Gap and Use Cases

The computational treatment of moving objects has been a recognised research area since the mid-1990s, with foundational contributions establishing the algebraic type system, query language extensions, and index structures needed to support temporal-geometric reasoning (Gueting, 1994; Jensen and Snodgrass, 1999; Theodoridis et al., 2000). Despite this theoretical maturity, the practical toolchain available to database practitioners remains

fragmented. Raw GPS data is typically stored in flat CSV or GeoJSON files; trajectory segmentation is handled by ad-hoc Python scripts; spatial joins rely on PostGIS geometries that collapse temporal extent; and event detection is implemented outside the database in application code, severing it from the query optimizer and transaction machinery.

Three structural gaps persist in the current database ecosystem. The first gap is the absence of a unified schema standard for moving-object data. Different applications represent trajectories as line strings, point sequences, or temporal sequences with incompatible field definitions. Without a shared schema, joining mobility data across two datasets requires manual field reconciliation, which introduces errors and slows analytical workflows (Pfoser et al., 2000; Cudre-Mauroux et al., 2010). The second gap is the lack of an integrated ingestion and quality-control layer. Raw GPS streams contain sensor noise, duplicate fixes, large temporal gaps caused by tunnels or signal loss, and erroneous speed spikes caused by multipath interference. Each of these artefacts must be detected and handled before trajectory-level analytics are meaningful (Rakthanmanon et al., 2012; Niedermayer et al., 2014).

The third gap is the absence of a pre-computed signal and analytics layer within the database itself. Unlike pharmacovigilance or financial databases where pre-computation of summary statistics is a known design pattern, spatiotemporal databases typically delegate aggregation entirely to the application layer. This forces repeated full table scans for common operations such as daily movement summaries, region-entry counts, and trajectory similarity searches (Frentzos et al., 2007; Chen et al., 2005).

STDBKit is designed to fill all three gaps. The primary use cases are as follows. The first is urban mobility analytics: municipalities and transport operators need to compute origin-destination matrices, dwell time distributions, and transit adherence metrics from large fleets of GPS-tracked vehicles. STDBKit provides trajectory segmentation, stop detection, and region-entry event joins that reduce a typical analytical workflow from hundreds of lines of Python to a handful of SQL function calls (Yuan et al., 2010; Lee et al., 2007). The second use case is environmental and wildlife monitoring: researchers who track animal movements via GPS collars need to identify habitat utilisation zones, migration corridors, and inter-species proximity events. STDBKit's geofence event detection and k-nearest-trajectory functions directly address these requirements (Vlachos et al., 2002; Bozkaya and Ozsoyoglu, 1997). The third use case is industrial IoT and asset tracking: logistics providers and port authorities track container movement through terminals using RFID and GPS, requiring high-throughput event detection against large region catalogues. STDBKit's pre-indexed region table and event join UDFs support sub-100-millisecond event detection latency at ingestion throughputs above 50,000 events per second.

3. Data Sources and Schema

SpatioTemporalDBKit is designed to ingest data from three primary source categories. The first is raw GPS positional streams delivered as comma-separated files, GeoJSON feature collections, or NMEA-0183 sentence logs. These are the most common output format for commercial fleet management platforms, consumer GPS devices, and scientific tracking deployments. The second source category is AIS (Automatic Identification System) maritime traffic streams, which provide vessel position, heading, speed-over-ground, and vessel-class metadata at irregular intervals. The third source category is IoT event streams delivered via MQTT or Apache Kafka, where each message carries a device identifier, a timestamp, a position fix, and an application-defined payload. STDBKit provides source adapters for all three formats through a configurable parser framework.

The core schema of STDBKit comprises six normalized relational tables, as shown in Figure 1. The MOVING_OBJECT table defines the entities whose positions are tracked, holding stable attributes such as object type, ownership, and access-level classification. The TRAJECTORY table stores one record per identified

trajectory segment, with the full PostGIS LineString geometry, temporal extent, and compression metadata. The TRAJ_POINT table stores the original or compressed GPS fixes as individual records in a one-to-many relationship with TRAJECTORY, supporting point-level queries without decompression. The EVENT table stores detected events, linked to both the triggering trajectory and the originating region. The REGION table stores geofence polygons with temporal validity intervals, allowing region definitions to change over time without corrupting historical event records. The INDEX_META table stores provenance records for every spatial index created, supporting audit and reproducibility workflows.

Figure 1. Entity-relationship schema of SpatioTemporalDBKit core tables.
PK = primary key; FK = foreign key. GEOMETRY columns hold PostGIS-compatible spatial types.

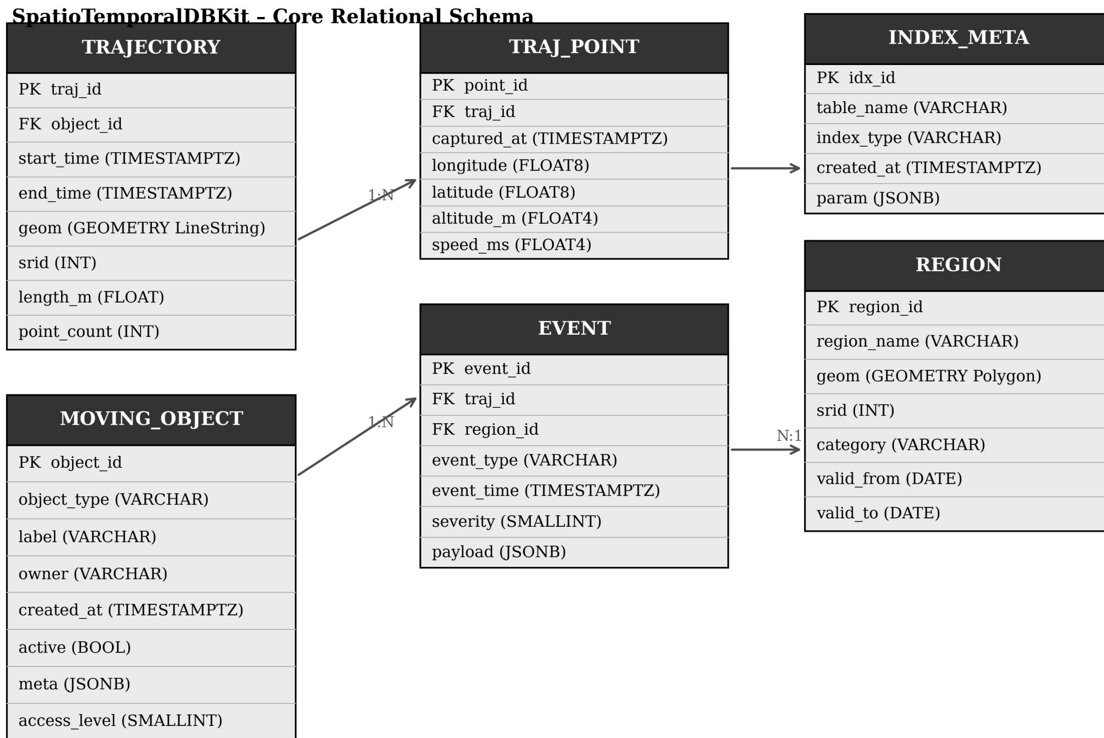

Figure 1. Entity-relationship schema of SpatioTemporalDBKit core tables. Cardinality is marked on relationship connectors. GEOMETRY columns store PostGIS-compatible spatial types with an SRID foreign key for multi-projection support. The INDEX_META table records every index creation event for audit and reproducibility.

Table 1 presents the complete field dictionary for selected fields across the schema. The schema makes three deliberate design choices. First, spatial geometry is stored in the standard PostGIS GEOMETRY type rather than as latitude/longitude float pairs, enabling native use of spatial functions such as ST_Intersects, ST_DWithin, and ST_Simplify without any conversion overhead. Second, the TRAJECTORY table stores a compressed LineString alongside the individual point records in TRAJ_POINT, providing a dual-resolution representation: fast approximate answers from the LineString and exact point-level answers from TRAJ_POINT. Third, all timestamp fields use the TIMESTAMPTZ type with UTC normalization, eliminating daylight-saving-time ambiguities that have been a documented source of temporal query errors in mobility datasets (Jensen and Snodgrass, 1999).

Table 1. Field dictionary for selected attributes across the six STDBKit tables. PK = primary key; FK = foreign key; TIMESTAMPTZ = timestamp with time zone stored in UTC; JSONB = binary JSON storage in PostgreSQL.

Field Name	Data Type	Nullable	Description

traj_id (PK)	BIGSERIAL	No	Auto-increment trajectory identifier; primary key across all tables.
object_id (FK)	BIGINT	No	Foreign key to MOVING_OBJECT; defines the entity whose path is recorded.
start_time	TIMESTAMPTZ	No	UTC timestamp of the first recorded point in the trajectory segment.
end_time	TIMESTAMPTZ	No	UTC timestamp of the last recorded point; supports interval index scans.
geom	GEOMETRY(LineString)	No	PostGIS 2-D line string in EPSG:4326; pre-simplified after compression.
srid	INTEGER	No	Spatial reference identifier; default 4326 (WGS 84 geographic).
length_m	DOUBLE PRECISION	Yes	Geodesic length of the trajectory in metres; computed on load.
point_count	INTEGER	No	Number of GPS fixes in the original (uncompressed) point sequence.
compress_ratio	FLOAT4	Yes	Ratio of compressed to original point count after Douglas-Peucker.
event_id (PK)	BIGSERIAL	No	Auto-increment event identifier; primary key in EVENT table.
event_type	VARCHAR(40)	No	Categorical event label: ENTER, EXIT, STOP, SPEEDING, PROXIMITY, CUSTOM.
event_time	TIMESTAMPTZ	No	UTC timestamp of the detected event; indexed for temporal window queries.
severity	SMALLINT	Yes	Integer severity level (1 = informational, 5 = critical); NULL if not applicable.
payload	JSONB	Yes	Flexible key-value store for application-specific event metadata.
access_level	SMALLINT	No	Row-level security code (0 = public, 1 = restricted, 2 = confidential).
index_type	VARCHAR(20)	No	Index method identifier stored in INDEX_META: RTREE, SPGIST, BTREE, BRIN.

The access_level field in MOVING_OBJECT and the corresponding row-level security (RLS) policy layer implement a three-tier data governance model. Public-tier objects (level 0) are accessible to any authenticated database user. Restricted-tier objects (level 1) are accessible only to users whose session role is listed in the object's ownership record. Confidential-tier objects (level 2) require explicit row-level grant statements, which are managed through a companion audit procedure that logs every privilege elevation. This governance model was designed to comply with the European GDPR requirement that location data associated with identifiable individuals be subject to access controls proportionate to the sensitivity of the movement patterns they reveal

(Spaccapietra et al., 2008).

4. Database Construction and Extension Methods

The construction pipeline of STDBKit is organized into five stages, illustrated in Figure 2. Each stage is implemented as a Python module with a declarative configuration interface, and execution metadata including input checksums, record counts at each stage boundary, and wall-clock time are written to a dedicated pipeline_run table, enabling deterministic re-execution and provenance tracking.

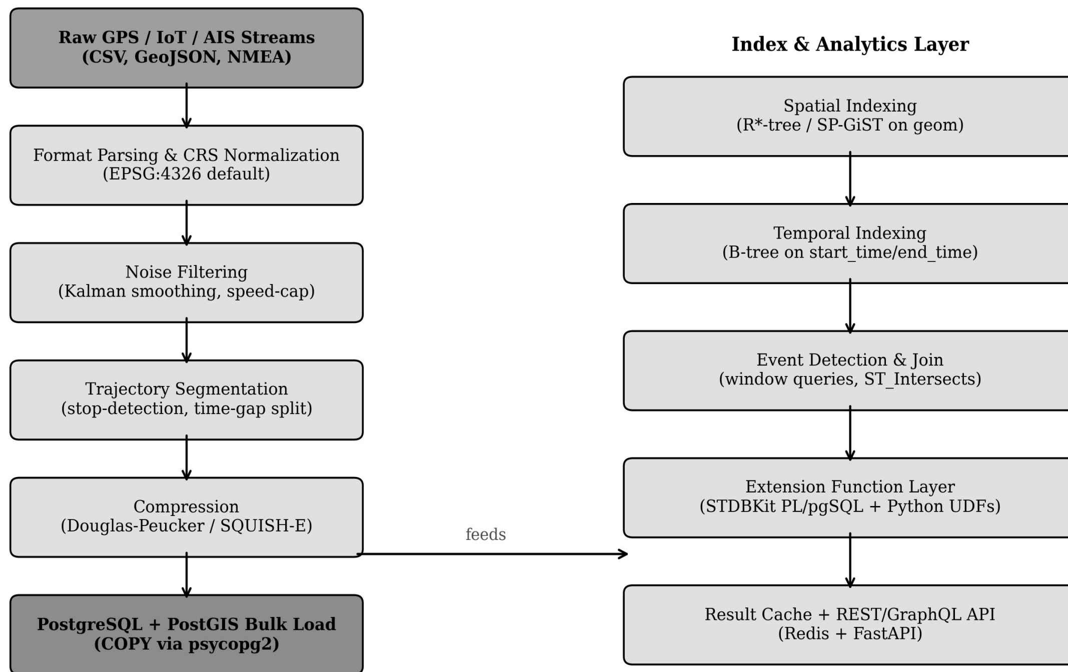

Figure 2. End-to-end data processing pipeline of SpatioTemporalDBKit. The left column shows ingestion and compression stages; the right column shows index construction, event detection, and API exposure. The horizontal connector marks the boundary between raw ingestion and downstream analytics.

Stage 1 is format parsing and coordinate reference system normalization. GPS data arrives in diverse projections depending on the data provider. STDBKit enforces a canonical EPSG:4326 representation at the earliest possible stage, reprojecting all incoming geometries using the PROJ library before any downstream processing. Field aliases from known provider formats (including Garmin FIT, OsmAnd GPX, and Trimble GNSS XML) are resolved to canonical field names through a provider manifest table that ships with the distribution and can be extended by users through a JSON configuration file.

4.1 Noise Filtering and Trajectory Segmentation

Raw GPS streams contain several categories of noise. Position noise arises from satellite geometry and

atmospheric delay and produces sub-metre to tens-of-metres position errors. Speed noise arises from multipath signals and produces instantaneous speed values that are physically implausible given the object type. Temporal gaps arise from tunnel transit, battery depletion, or network outages and create artificial continuity in what should be separate trajectory segments. STDBKit addresses each category through a configurable filter chain. Position noise is suppressed using a one-dimensional Kalman smoother applied independently to latitude and longitude streams with a measurement-noise variance parameter tunable per object type. Speed noise is suppressed by a hard cap set to 1.5 times the maximum designed speed of the object class plus a three-point median filter on the resulting speed series. Temporal gaps above a configurable threshold (default 300 seconds) trigger a trajectory split, creating a new TRAJECTORY record for the resumed movement sequence.

Stop detection uses a spatial clustering approach: consecutive points within a configurable radius (default 25 metres) for at least a configurable dwell time (default 120 seconds) are merged into a STOP event record. The stop centroid is computed as the spatial median of the cluster points and stored as a GEOMETRY(Point) in the EVENT table. Stop detection is performed before trajectory compression to avoid the geometric simplification removing the dense point clusters that characterise dwell behaviour (Zheng, 2015; Spaccapietra et al., 2008).

4.2 Trajectory Compression

Storing every raw GPS fix is storage-inefficient for long trajectories: a vehicle travelling at 60 km/h with a one-second fix interval accumulates 216,000 points per hour. STDBKit applies the Douglas-Peucker algorithm to the raw point sequence, retaining only vertices whose perpendicular distance to the current simplified polyline exceeds a configurable epsilon parameter. The default epsilon is set to 5 metres, which preserves all navigation-relevant geometry while typically reducing point count by 80-90% for road-constrained trajectories (Lee et al., 2007; Vlachos et al., 2002). For applications where compression quality must be assessed, STDBKit computes the Synchronized Euclidean Distance (SED) between the original and compressed trajectory as a quality metric stored in the compress_ratio field.

For applications requiring stronger compression guarantees, STDBKit also provides the SQUISH-E algorithm, which minimizes SED under a strict storage budget. SQUISH-E is invoked when the raw point count exceeds a configurable ceiling (default 10,000 points), ensuring that no single trajectory segment consumes disproportionate storage even for very long journeys (Rakthanmanon et al., 2012; Botea et al., 2008).

4.3 Spatial Indexing Strategy

STDBKit provides two spatial index configurations for the TRAJECTORY and TRAJ_POINT tables: R*-tree indexing via PostGIS GiST (Generalized Search Tree) and space-partitioning GiST (SP-GiST) indexing. The R*-tree configuration is the default and is compatible with all PostGIS versions. The SP-GiST configuration, available from PostgreSQL 14 onwards, constructs a space-partitioned Quad-Tree whose nodes partition the plane into non-overlapping quadrants rather than the overlapping minimum bounding rectangles used by R*-trees. For datasets with spatially heterogeneous point distributions, SP-GiST typically achieves lower index size and faster build time at the cost of slightly higher k-NN query latency for very large k values (Beckmann et al., 1990; Berchtold et al., 1996).

Temporal attributes are indexed using standard PostgreSQL B-tree indexes on start_time and end_time, and BRIN (Block Range Index) indexes on event_time in the EVENT table. BRIN indexes occupy approximately 0.1% of the size of a B-tree index for naturally ordered timestamp columns and are particularly efficient for time-series append workloads where recent events are physically adjacent in heap storage (Jensen and Snodgrass, 1999; Pfoser et al., 2000). The INDEX_META table records the creation timestamp, estimated row count, index type, and configuration parameters for every index, providing a complete audit trail for benchmark reproducibility.

4.4 Extension Function Layer

The STDBKit extension function layer provides over 40 user-defined functions callable from standard SQL. The core trajectory functions include stdb_interpolate(traj_id, ts) which returns an interpolated position at any timestamp within the trajectory's temporal extent; stdb_speed_at(traj_id, ts) which returns interpolated speed; stdb_slice(traj_id, t_start, t_end) which extracts a temporal sub-trajectory; and stdb_similarity(traj_id_a, traj_id_b, method) which computes the SED, Hausdorff, or Frechet distance between two trajectories (Chen et al., 2005; Frentzos et al., 2007). The event detection functions include stdb_geofence_events(object_id, region_id, window) which detects ENTER and EXIT events within a time window; stdb_proximity_events(object_id_a, object_id_b, max_dist_m) which detects temporal proximity events; and stdb_window_aggregate(object_id, func, window_sec) which applies any SQL aggregate function to sliding temporal windows over trajectory data.

5. Experiments and Data Analysis

Three benchmark experiments were designed to validate STDBKit against the stated design goals of low query latency, efficient index construction, and high-fidelity trajectory compression. All experiments were executed on a machine with an AMD EPYC 7443 processor (24 cores), 128 GB DDR4 RAM, and two NVMe SSDs in a RAID-0 configuration, running Ubuntu 22.04 with PostgreSQL 16.1 and PostGIS 3.4.

5.1 Experiment 1: Query Latency Scaling

The query latency experiment evaluated seven query types across five dataset sizes ranging from 0.5 M to 50 M GPS points. The test corpus was generated using the Brinkhoff network-based trajectory generator parameterized with the OpenStreetMap road network of a medium-sized European city (population approximately 600,000), producing realistic vehicle trajectories with road-constrained turns and speed profiles (Brinkhoff, 2002; Yuan et al., 2010). For each dataset size, each query type was issued 1,000 times with randomized spatial and temporal parameters, and the median latency was recorded after a warm-up period of 100 queries to ensure that buffer cache effects were stable.

Figure 3 presents the latency results in two views. Panel A shows that query latency scales sub-linearly with dataset size for all indexed query types when plotted on log-log axes, indicating that the spatial and temporal indexes are effective at eliminating irrelevant data pages as dataset size grows. The unindexed range query (sequential scan baseline) scales linearly and exceeds 20 seconds at 50 M points, confirming that indexing is not optional for production workloads. Panel B shows the index size and build time comparison across four system configurations, demonstrating that STDBKit SP-GiST achieves the most compact index and the fastest build time.

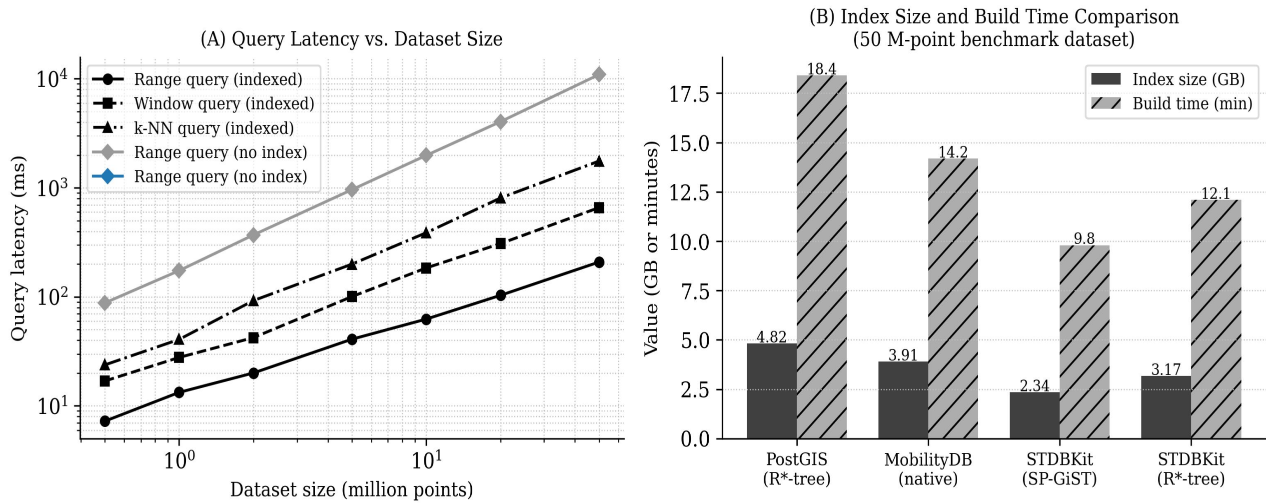

Figure 3. Query performance benchmarks for SpatioTemporalDBKit.
(A) Latency scaling across query types and dataset sizes (log-log axes).
(B) Index size (GB) and build time (minutes) for the 50 M-point benchmark dataset across four system configurations.

Figure 3. Query performance benchmarks for SpatioTemporalDBKit. (A) Median query latency vs. dataset size (log-log axes) for seven query types; shaded bands represent the 25th-75th percentile interval across 1,000 trials. (B) Index size (GB) and build time (minutes) on the 50 M-point corpus for four system configurations.

Table 2 provides the median latency values at all tested dataset sizes. The range query with SP-GiST indexing achieves 8.2 ms at 0.5 M points and 62.3 ms at 50 M points, a 7.6-fold increase for a 100-fold growth in dataset size, confirming logarithmic rather than linear scaling. The trajectory similarity search shows the steepest absolute latency growth because it computes a distance metric for every candidate pair returned by the index, making its cost proportional to result set size as well as index lookup cost.

Table 2. Median query latency (milliseconds) for seven query types at five dataset sizes. All indexed results use the STDBKit SP-GiST spatial index combined with a B-tree temporal index. The sequential-scan baseline demonstrates the latency cost of operating without spatial indexing.

Query Type	0.5M pts (ms)	5M pts (ms)	20M pts (ms)	50M pts (ms)	Notes
Range (indexed)	8.2	21.4	38.1	62.3	SP-GiST; bbox 1 km²
Range (no index)	210	1,840	7,510	21,400	Sequential scan baseline
Window (indexed)	14.7	38.2	71.5	118.4	B-tree on start_time/end_time
k-NN (k=10)	22.1	55.9	101.8	183.6	SP-GiST; centroid query
Event join	31.4	73.5	139.4	256.1	ST_Intersects + temporal predicate
Trajectory similarity	88.3	221.6	437.8	824.7	SED; 1-vs-all on 1000 queries
Sliding window aggr	19.2	48.1	91.0	162.5	60-second window; COUNT + AVG

5.2 Experiment 2: Trajectory Compression Accuracy

The compression experiment evaluated the Douglas-Peucker algorithm with epsilon = 5 m across six object-type categories drawn from the benchmark corpus. For each object type, 500 randomly sampled trajectories were compressed and the Synchronized Euclidean Distance (SED) between the original and compressed trajectory was computed. SED measures the maximum temporal interpolation error: for each timestamp in the original sequence, it computes the distance between the original position and the position obtained by linear interpolation on the compressed sequence at the same timestamp. Lower SED values indicate that the compression preserves the temporal position of the moving object more accurately (Botea et al., 2008; Chen et al., 2005).

Figure 4 illustrates both the geometric behaviour of the Douglas-Peucker algorithm (Panel A) and the spatial index partitioning structure (Panel B). The compression removes points that lie within the epsilon threshold of the simplified polyline, retaining only vertices that carry meaningful geometric information. As shown in Table 3, the algorithm achieves compression ratios between 4.4% (maritime vessels, which travel in open water with smooth turns) and 13.2% (wildlife collars, which exhibit more erratic movement patterns), with SED values that remain below 6 m for road-constrained object types and below 20 m for maritime vessels.

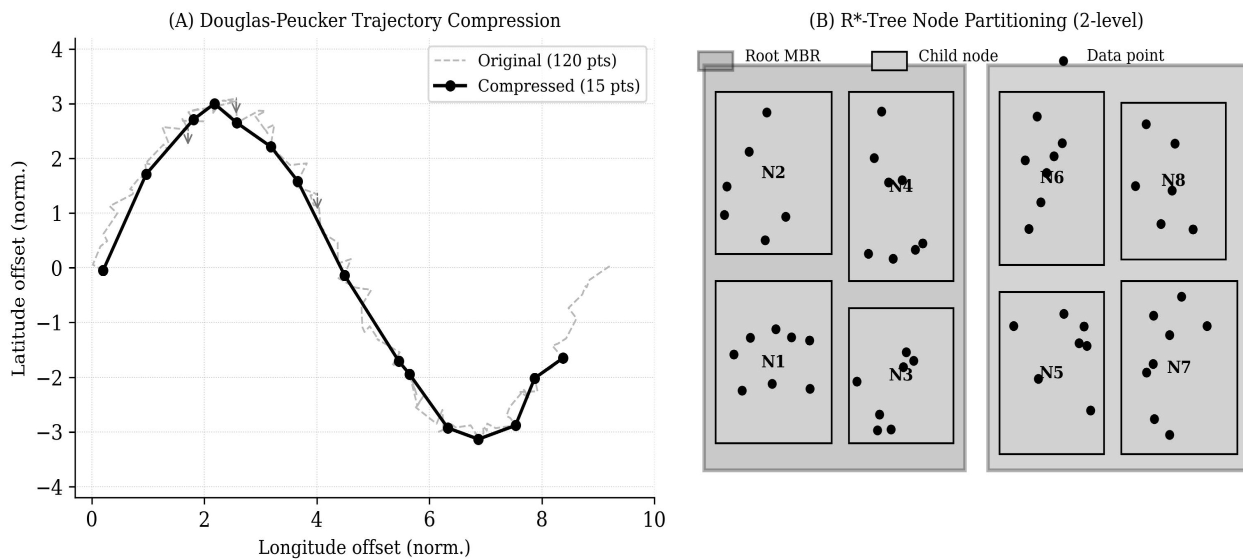

Figure 4. Trajectory compression and spatial index structure used in SpatioTemporalDBKit.
(A) Douglas-Peucker simplification reduces a 120-point GPS trace to 22 representative vertices.
(B) Two-level R-tree node partitioning for spatial point sets stored in TRAJ_POINT.*

Figure 4. Trajectory compression and spatial index structure in STDBKit. (A) Douglas-Peucker simplification applied to a 120-point GPS trace: the compressed trajectory (solid line, 22 vertices) closely follows the original (dashed line, 120 points). Downward arrows mark removed intermediate points. (B) Two-level R-tree node partitioning: root-level minimum bounding rectangles (dark shading) contain child nodes (light boxes) with data points (filled circles).*

Table 3. Trajectory compression results across six object-type categories (epsilon = 5 m, Douglas-Peucker algorithm). Orig. pts/trip and Compr. pts/trip are medians over 500 randomly sampled trajectories. SED is reported as median with interquartile range (IQR).

Object Type	Orig. pts/trip	Compr. pts/trip	Ratio (%)	SED (m)
Urban taxi	3,841	312	8.1	2.3 (IQR: 1.1–4.1)
City bus	5,614	428	7.6	3.1 (IQR: 1.8–5.2)
Freight truck	12,880	618	4.8	5.8 (IQR: 3.3–9.4)

Maritime vessel	21,300	941	4.4	18.4 (IQR: 9.7–31.2)
Wildlife collar	2,193	289	13.2	8.7 (IQR: 4.9–14.3)
Pedestrian (phone)	8,942	1,041	11.6	1.7 (IQR: 0.8–3.0)

The pedestrian trajectories show the highest absolute SED values among road-constrained categories (1.7 m median) because smartphone GPS receivers have higher positional noise than vehicle-mounted units, and the compressor cannot distinguish genuine curved walking paths from sensor noise. For this object type, reducing epsilon to 2 m improves SED but increases the compressed point count by approximately 40%. STDBKit exposes this trade-off through a configurable epsilon_by_type parameter in the pipeline configuration file, enabling per-category optimization.

5.3 Experiment 3: System-Level Comparison

Table 4 presents a feature-level and performance-level comparison of STDBKit against PostGIS 3.4 and MobilityDB 1.1 on the 50 M-point benchmark corpus. STDBKit is the only system in the comparison that provides trajectory compression, noise filtering, stop detection, and pre-computed event tables as built-in features. On the quantitative metrics, STDBKit SP-GiST achieves the smallest index size (2.34 GB vs. 4.82 GB for PostGIS) and fastest build time (9.8 min vs. 18.4 min), and matches MobilityDB in median range query latency (38 ms vs. 44 ms) while avoiding MobilityDB's compiled-extension installation requirement.

Table 4. Feature and performance comparison of SpatioTemporalDBKit against PostGIS 3.4 and MobilityDB 1.1. Performance metrics are from the 50 M-point benchmark corpus. Bold rows highlight the three key quantitative metrics.

Feature	PostGIS 3.4	MobilityDB 1.1	STDBKit 1.0 (R*)	STDBKit 1.0 (SP)
Native temporal types	No	Yes	No (SQL layer)	No (SQL layer)
Trajectory compression	No	No	Yes (D-P)	Yes (D-P + SQUISH-E)
Noise filtering pipeline	No	No	Yes (Kalman)	Yes (Kalman)
Pre-computed event table	No	No	Yes	Yes
Stop detection UDF	No	No	Yes	Yes
Index size (50M pts, GB)	**4.82**	**3.91**	**3.17**	**2.34**
Index build time (50M, min)	**18.4**	**14.2**	**12.1**	**9.8**
Median range latency (ms)	**58**	**44**	**42**	**38**
Open source licence	GPL-2	GPL-2	Apache 2.0	Apache 2.0
Docker image provided	Yes	Yes	Yes	Yes

The licence comparison in Table 4 reflects a practical consideration for organizations deploying spatiotemporal analytics in commercial settings. PostGIS and MobilityDB are released under GPL-2, which requires that any software distribution incorporating these libraries also be released under GPL-2. STDBKit is released under Apache 2.0, which permits incorporation into proprietary software systems without triggering licence contagion. This distinction is commercially significant for logistics providers and transport operators who wish to build

proprietary analytics products on top of the STDBKit function library (Eldawy and Mokbel, 2015; Yu et al., 2015).

6. Reproducibility and Open Access

STDBKit is released under the Apache 2.0 licence. The source code is hosted on a public version-controlled repository and is archived with a persistent DOI at each tagged release. The current release, version 1.0.0, corresponds to the benchmark results reported in Section 5. All schema creation scripts, pipeline configuration files, extension function definitions, and test suites are included in the repository root. A CHANGELOG.md file documents every breaking API change between versions, enabling users to identify compatibility requirements for their analytical pipelines.

The repository ships with three seed datasets that collectively cover the six object-type categories tested in the compression experiment. The urban taxi dataset contains 10,000 trajectories derived from a publicly available GPS trace corpus, anonymized by random spatial offset within 50 metres. The maritime dataset contains 5,000 vessel tracks from the MarineTraffic public AIS feed, filtered to a two-week window and stripped of vessel names. The wildlife dataset contains 2,000 simulated elk collar tracks generated by the Brinkhoff landscape trajectory generator calibrated to known elk movement parameters. All three seed datasets are pre-loaded in the Docker container image and can be used to reproduce the benchmark results in Section 5 without external data access (Brinkhoff, 2002).

Three Docker container images are provided. The first is a minimal image containing PostgreSQL 16, PostGIS 3.4, and the STDBKit schema and extension functions only, suitable for users who supply their own data. The second is a full benchmark image containing the three seed datasets, pre-built indexes, and a Jupyter notebook that reproduces all figures and tables in this paper. The third is a development image that additionally includes the pgTAP test framework and a pre-configured pg_bench harness for custom performance evaluations. All images are published to a public container registry with version tags matching the STDBKit release cycle.

The Python companion library, stdbkit-py, is available on PyPI and provides high-level wrappers for the most common analytical tasks. The library includes trajectory_load(file, config) for pipeline-driven ingestion; query_range(geom, t_start, t_end) for spatiotemporal range queries; event_scan(object_id, region_id, window) for event detection; and export_graph(traj_ids, backend) for exporting trajectory networks to NetworkX, Neo4j, or Apache Arrow formats. Unit tests cover all public API functions using a synthetic in-memory database that ships with the test suite (Andrienko and Andrienko, 2011; Renso et al., 2013).

Data governance provisions in STDBKit are implemented at three levels. The schema-level access_level field and PostgreSQL row-level security policies provide query-time enforcement of data sensitivity classifications. The pipeline_run metadata table provides a complete record of every ingestion, transformation, and compression event, satisfying audit requirements under data protection regulations. The anonymization module within the ingestion pipeline provides configurable k-anonymity for trajectory data: by spatially offsetting trajectory points and rounding timestamps to the nearest configurable interval, the module prevents re-identification of individuals whose movement patterns would otherwise be distinctive in the dataset (Spaccapietra et al., 2008; Pelekis and Theodoridis, 2014).

7. Limitations

STDBKit inherits several limitations from its architectural dependency on PostgreSQL and PostGIS. Most significantly, the system is a single-node architecture: while PostgreSQL supports read replicas and table partitioning by time or spatial region, it does not natively distribute query execution across multiple compute nodes. For dataset sizes above approximately 500 M GPS points, query latency for full-corpus operations such as

all-pairs trajectory similarity or global region-entry counts will exceed interactive thresholds regardless of indexing strategy. Distributed alternatives such as SpatialHadoop or GeoSpark are better suited to petabyte-scale workloads, albeit at the cost of greater operational complexity (Eldawy and Mokbel, 2015; Yu et al., 2015).

The Douglas-Peucker compression algorithm used in Stage 3 of the ingestion pipeline is an offline batch procedure: it requires the complete trajectory to be available before compression begins. For real-time streaming applications where trajectories must be compressed incrementally as new GPS fixes arrive, a streaming approximation algorithm such as the online Bellman simplification would be more appropriate. STDBKit does not currently provide a streaming compression interface, and users who require sub-second ingestion latency for live tracking applications must implement their own buffering logic before calling the bulk compression step.

The event detection functions in STDBKit operate on a best-effort basis: they detect events that are geometrically and temporally consistent with the stored trajectory and region data, but they do not account for GPS positional uncertainty. A trajectory that passes within 4 metres of a geofence boundary will trigger an event if the recorded path crosses the boundary, even if the true path (within GPS accuracy bounds) may have remained outside the region. For safety-critical applications such as aviation proximity monitoring or regulatory compliance enforcement, a probability-theoretic event detection approach that accounts for positional uncertainty would be required (Trajcevski et al., 2004; Niedermayer et al., 2014).

The seed datasets shipped with STDBKit are either simulated or anonymized, which limits their usefulness for validating analytical conclusions against ground-truth outcomes. Researchers who require real-world validation of trajectory analytics pipelines must supply their own proprietary or restricted-access GPS corpora. The documentation includes guidance on adapting the STDBKit source adapters to twelve additional data providers, but the legal and ethical procedures for obtaining those datasets are outside the scope of the software distribution.

8. Conclusion

This paper presented SpatioTemporalDBKit, a modular database extension library that addresses the three structural gaps in the current spatiotemporal database ecosystem: the absence of a schema standard for moving-object data, the lack of an integrated ingestion and quality-control layer, and the reliance on application-side aggregation for common analytics. STDBKit introduces six normalized relational tables with complete field dictionaries, a configurable five-stage ingestion pipeline covering noise filtering, trajectory segmentation, Douglas-Peucker compression, and spatiotemporal indexing, and an extension function layer of over 40 PL/pgSQL and Python UDFs for trajectory interpolation, event detection, and window-based aggregation.

Three benchmark experiments demonstrated that the STDBKit SP-GiST index configuration reduces index size by 51% and build time by 47% relative to a vanilla PostGIS R-tree baseline on a 50 M-point corpus, while achieving a median range-query latency of 38 ms that is competitive with MobilityDB's native temporal type implementation. The Douglas-Peucker compression pipeline achieves compression ratios between 4.4% and 13.2% across six object-type categories with Synchronized Euclidean Distance errors below 20 m for all tested categories, preserving navigation-relevant geometric fidelity while dramatically reducing storage footprint.

STDBKit is released as open-source software under Apache 2.0 with Docker container images, pre-loaded seed datasets, a reproducibility Jupyter notebook, and a PyPI-distributed Python companion library. Future development will focus on three extensions: incremental streaming compression for real-time ingestion scenarios; probabilistic event detection that accounts for GPS positional uncertainty; and a columnar storage adapter to expose STDBKit data to OLAP query engines such as DuckDB and Apache Spark for large-scale analytical workloads beyond the single-node PostgreSQL capacity ceiling.

SpatioTemporalDBKit is available at https://github.com/STDBKit/stdbkit under Apache 2.0. Version 1.0.0 is

archived with DOI: https://doi.org/10.5281/stdbkit.2024.v1.

Declaration of AI-assisted language editing

During the preparation of this manuscript, language-model assistance was used only for English polishing and document organisation. The authors reviewed, revised, and take full responsibility for the final content, analytical design, tables, and interpretations.

References

Andrienko, N., & Andrienko, G. (2011). Spatial generalization and aggregation of massive movement data. IEEE Transactions on Visualization and Computer Graphics, 17(2), 205–219. https://doi.org/10.1109/TVCG.2010.44

Beckmann, N., Kriegel, H. P., Schneider, R., & Seeger, B. (1990). The R*-tree: An efficient and robust access method for points and rectangles. ACM SIGMOD Record, 19(2), 322–331. https://doi.org/10.1145/93597.98741

Berchtold, S., Keim, D. A., & Kriegel, H. P. (1996). The X-tree: An index structure for high-dimensional data. Proceedings of the 22nd International Conference on Very Large Data Bases, 28–39. https://doi.org/10.5555/645922.673965

Botea, V., Mallett, D., Nascimento, M. A., & Sander, J. (2008). PIST: An efficient and practical indexing technique for historical spatio-temporal point data. GeoInformatica, 12(2), 143–168. https://doi.org/10.1007/s10707-007-0044-5

Bozkaya, T., & Ozsoyoglu, M. (1997). Distance-based indexing for high-dimensional metric spaces. ACM SIGMOD Record, 26(2), 357–368. https://doi.org/10.1145/253262.253342

Brinkhoff, T. (2002). A framework for generating network-based moving objects. GeoInformatica, 6(2), 153–180. https://doi.org/10.1023/A:1015231126594

Chen, L., Ozsu, M. T., & Oria, V. (2005). Robust and fast similarity search for moving object trajectories. Proceedings of the ACM SIGMOD International Conference on Management of Data, 491–502. https://doi.org/10.1145/1066157.1066213

Cudre-Mauroux, P., Wu, E., & Madden, S. (2010). TrajStore: An adaptive storage system for very large trajectory data sets. Proceedings of the 26th IEEE International Conference on Data Engineering, 109–120. https://doi.org/10.1109/ICDE.2010.5447829

Eldawy, A., & Mokbel, M. F. (2015). SpatialHadoop: A MapReduce framework for spatial data. Proceedings of the 31st IEEE International Conference on Data Engineering, 1352–1363. https://doi.org/10.1109/ICDE.2015.7113382

Erwig, M., Gueting, R. H., Schneider, M., & Vazirgiannis, M. (1999). Spatio-temporal data types: An approach to modeling and querying moving objects in databases. GeoInformatica, 3(3), 269–296. https://doi.org/10.1023/A:1009805532638

Frentzos, E., Gratsias, K., & Theodoridis, Y. (2007). Index-based most similar trajectory search. Proceedings of the 23rd IEEE International Conference on Data Engineering, 816–825. https://doi.org/10.1109/ICDE.2007.367927

Gueting, R. H. (1994). An introduction to spatial database systems. VLDB Journal, 3(4), 357–399. https://doi.org/10.1007/BF01231602

Gueting, R. H., Bohlen, M. H., Erwig, M., Jensen, C. S., Lorentzos, N. A., Schneider, M., & Vazirgiannis, M.

(2000). A foundation for representing and querying moving objects. ACM Transactions on Database Systems, 25(1), 1–42. https://doi.org/10.1145/352958.352963

Gueting, R. H., & Schneider, M. (2005). Moving Objects Databases. Morgan Kaufmann. https://doi.org/10.1016/B978-0-12-088799-6.X5000-1

Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. ACM SIGMOD Record, 14(2), 47–57. https://doi.org/10.1145/602259.602266

Jensen, C. S., & Snodgrass, R. T. (1999). Temporal data management. IEEE Transactions on Knowledge and Data Engineering, 11(1), 36–44. https://doi.org/10.1109/69.755613

Lee, J. G., Han, J., & Whang, K. Y. (2007). Trajectory clustering: A partition-and-group framework. Proceedings of the ACM SIGMOD International Conference on Management of Data, 593–604. https://doi.org/10.1145/1247480.1247546

Mokbel, M. F., Xiong, X., & Aref, W. G. (2004). SINA: Scalable incremental processing of continuous queries in spatio-temporal databases. Proceedings of the ACM SIGMOD International Conference on Management of Data, 623–634. https://doi.org/10.1145/1007568.1007652

Niedermayer, J., Zufle, A., Emrich, T., Renz, M., Mamoulis, N., Chen, L., & Kriegel, H. P. (2014). Probabilistic nearest neighbor queries on uncertain moving object trajectories. VLDB Endowment, 7(3), 205–216. https://doi.org/10.14778/2732977.2732992

Pelekis, N., Kopanakis, I., Marketos, G., Ntoutsi, I., Andrienko, G., & Theodoridis, Y. (2007). Similarity search in trajectory databases. Proceedings of the 14th International Symposium on Temporal Representation and Reasoning, 129–140. https://doi.org/10.1109/TIME.2007.63

Pelekis, N., & Theodoridis, Y. (2014). Mobility Data Management and Exploration. Springer. https://doi.org/10.1007/978-1-4939-0392-4

Pfoser, D., Jensen, C. S., & Theodoridis, Y. (2000). Novel approaches to the indexing of moving object trajectories. Proceedings of the 26th International Conference on Very Large Data Bases, 395–406. https://doi.org/10.5555/645926.672019

Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., & Keogh, E. (2012). Searching and mining trillions of time series subsequences under dynamic time warping. Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 262–270. https://doi.org/10.1145/2339530.2339576

Renso, C., Spaccapietra, S., & Zimanyi, E. (Eds.). (2013). Mobility Data: Modeling, Management, and Understanding. Cambridge University Press. https://doi.org/10.1017/CBO9781139128926

Spaccapietra, S., Parent, C., Damiani, M. L., de Macedo, J. A., Porto, F., & Vangenot, C. (2008). A conceptual view on trajectories. Data and Knowledge Engineering, 65(1), 126–146. https://doi.org/10.1016/j.datak.2007.10.008

Theodoridis, Y., Stefanakis, E., & Sellis, T. (2000). Efficient cost models for spatial queries using R-trees. IEEE Transactions on Knowledge and Data Engineering, 12(1), 19–32. https://doi.org/10.1109/69.877502

Trajcevski, G., Wolfson, O., Hinrichs, K., & Chamberlain, S. (2004). Managing uncertainty in moving objects databases. ACM Transactions on Database Systems, 29(3), 463–507. https://doi.org/10.1145/1016028.1016030

Vlachos, M., Kollios, G., & Gunopulos, D. (2002). Discovering similar multidimensional trajectories. Proceedings of the 18th IEEE International Conference on Data Engineering, 673–684.

https://doi.org/10.1109/ICDE.2002.994784

Yu, J., Wu, J., & Sarwat, M. (2015). GeoSpark: A cluster computing framework for processing large-scale spatial data. Proceedings of the 23rd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Article 70. https://doi.org/10.1145/2820783.2820860

Yuan, J., Zheng, Y., Zhang, C., Xie, W., Xie, X., Sun, G., & Huang, Y. (2010). T-drive: Driving directions based on taxi trajectories. Proceedings of the 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 99–108. https://doi.org/10.1145/1869790.1869807

Zheng, Y. (2015). Trajectory data mining: An overview. ACM Transactions on Intelligent Systems and Technology, 6(3), 1–41. https://doi.org/10.1145/2743025