Performance Benchmarking of Geospatial Data Formats for Storing and Querying Agricultural Data

Saša Brdnik[1,*], Luka Prepadnik[1] and Muhamed Turkanović[1]

[1]University of Maribor Faculty of Electrical Engineering and Computer Science Koroška cesta 46, SI-2000 Maribor, Slovenia

Abstract

The choice of vector data format has a direct impact on storage and performance in geospatial workflows, yet its selection is often impacted by legacy choices and convenience. This work reports the results of a controlled experiment benchmarking four geospatial formats, GeoParquet, GeoJSON, CSV, and GeoPackage, and two geospatial data processing tools (GeoPandas and DuckDB). Four performance scenarios were observed: i) reading (loading) the file, ii) writing a file, iii) reading, appending, and writing the file, and iv) query execution. Evaluation was conducted using the EuroCrops 2021 dataset for Slovenia (828,154 agricultural parcels, 12 attributes, 8 million coordinates). The results indicate that the format is the primary factor in performance, while the choice of the geospatial data processing tool is secondary. GeoParquet outperformed other formats by a large margin in all scenarios except the most geometry-heavy query, where format differences largely vanished. GeoJSON was consistently the slowest, largest, and thus impractical at this scale. DuckDB was generally faster and introduced less variability across runs than GeoPandas, though its advantage narrowed for geometry-bound operations. Across all formats, appending forced a full-file rewrite, so incremental-update pipelines should favor batch writes as large as possible.

Keywords

GeoParquet, GeoJSON, GeoPackage, CSV, benchmark, EuroCrops dataset, geospatial data

1. Introduction

The amount of generated geospatial data has been steadily increasing in recent years. In Europe alone, the Copernicus Data Space Ecosystem held nearly 80 petabytes (PB) of online Earth observation data and over 100 million Sentinel products by the end of 2024, and made more than 200 PB of data available to users over the course of that year [1]. NASA's Earth science data archive surpassed 123 PB by the end of the same year [2]. To follow the analytical requirements of the increasingly available data, the landscape of geodata formats and GIS systems has been rapidly changing. Columnar data formats such as Spatial Parquet [3] and GeoParquet [4] were introduced to meet these requirements, offering efficient compression, reduced storage and I/O overhead, and faster analytical query execution. While much of this growth is driven by raster Earth observation imagery, vector geospatial data is expanding in parallel. OpenStreetMap, which is the largest crowdsourced vector geospatial database, has grown rapidly and continuously since its inception and now includes billions of geometric primitives (nodes, ways, and relations) [5].

Furthermore, administrative and survey-based sources, including parcels, building footprints, and the agricultural parcel declarations that EU Member States collect annually under the common agricultural policies, are increasingly published in the form of large, openly available datasets. The EuroCrops collection, which harmonizes self-declared crop parcels across the European Union, exemplifies this trend, comprising hundreds of thousands of polygons per country [6]. Efficiently storing, querying, and updating such vector datasets is therefore an increasingly common requirement.

Spatial systems and libraries available are diverse and may offer limited support for the increasing

SQAMIA 2026, September 07–11, 2026, Niš, Serbia

*Corresponding author.

†These authors contributed equally.

sasa.brdnik@um.si (S. Brdnik); luka.prepadnik@student.um.si (L. Prepadnik); muhamed.turkanovic@um.si (M. Turkanović)

0000-0003-3730-2769 (S. Brdnik); 0000-0002-5079-5468 (M. Turkanović)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-19

CEUR-WS.org/Vol-4336/paper13.pdf

number of data formats. This raises the challenge of selecting the best tool for each project's goals and needs. In practice, working with vector data covers a full lifecycle. Data must be written to disk, read back into memory, incrementally updated as new records arrive, and queried through filters, aggregations, and spatial joins. Existing format comparisons, however, focus almost exclusively on read (loading) performance, leaving write, incremental-insert, and query behavior largely unexplored. The relative performance of a format also depends on the processing engine, as a traditional in-memory dataframe library, such as GeoPandas, and an embedded analytical database, such as DuckDB, incorporate different execution paradigms and may favor different formats and operations.

In this work, we present a systematic benchmark of four representative geospatial formats, namely GeoParquet, GeoPackage, GeoJSON, and CSV, across two tools, GeoPandas and DuckDB, using the subset of the real-world EuroCrops dataset (data for Slovenia in 2021) [7]. Focusing on the vector dataset, this work aims to answer the following research questions:

RQ1: How do the data formats compare in data *storage size*?

RQ2: How do the data formats compare in data *read speed*?

RQ3: How do the data formats compare in data *write speed*? (INSERT rows [100, 1000, 10000], WRITE file (saving))

RQ4: How do read, query, and write speeds compare across different geospatial data tools? (DuckDB, GeoPandas)

2. Related work

Prior benchmarking work in geospatial storage formats has mostly focused on read (loading) throughput, and on a narrow range of data types and workloads. Middleton et al. [8] benchmarked GeoParquet against CSV, JSON, GeoJSON, and GeoPackage for land cover classification using Sentinel-2 imagery and ESA WorldCover labels. They evaluated them at the city and state scales in New Hampshire. Their analysis was limited to raster data, leaving the behavior of observed formats on vector geometries unexamined. Wachs and Zacharatou [9] have compared GeoParquet, GeoJSON, Shapefile and CSV across three geospatial data processing tools: DuckDB, OGR, and GeoPandas. Their findings suggest that GeoParquet delivers the highest loading throughput across all observed formats. Their evaluation was limited to the loading operation and was conducted on two types of datasets: first on synthetic vector datasets and then on a single real-world dataset (OpenStreetMap data for Denmark). In a related study, Saeedan and Eldawy [3] propose Spatial Parquet, a columnar file format for geospatial data lakes. Authors have benchmarked it against GeoParquet, Shapefile, and GeoJSON, reporting substantial reductions in storage size with delta encoding. The focus of this work is on a new storage encoding rather than on the read, write, and insert behavior of established formats. At the infrastructure level, OGC engineering report [10] compared and optimized GeoParquet alongside GeoZarr and Cloud-Optimized GeoTIFF (COG) for high-performance computing, focusing on bridging cloud-native geospatial standards with traditional supercomputing architectures, rather than on application-level read and write performance. Broader related work has also focused on benchmarking spatial query engines rather than storage formats. Pandey et al. [11] evaluated five Spark-based distributed spatial analytics systems (GeoSpark, Simba, SpatialSpark, Magellan, and LocationSpark) on real-world OpenStreetMap data, while GeoYCSB [12] provided a framework for performance and scalability evaluation in NoSQL databases on geospatial workloads. Beyond the peer-reviewed literature, practitioner explorations [13] also confirm the general advantage of columnar data formats but do not provide systematic, reproducible benchmarks.

Several research gaps can be observed in the existing literature. First, existing benchmarks focus almost exclusively on read (loading) performance. The cost of analytical read operations, such as spatial joins and group-by aggregation, directly over these formats has not been analyzed (RQ2). Second, write performance is largely unexamined, and to the best of our knowledge, no study evaluates incremental insert workloads, in which records are appended to an existing file in batches of increasing size (e.g.,

100, 1000, and 10,000 rows), despite this being a common pattern in operational data pipelines (RQ3). Third, although individual libraries have been compared for loading [9], the relative read, query, and write performance of DuckDB and GeoPandas across formats has not yet been systematically analyzed (RQ4). Finally, the datasets used are predominantly synthetic or derived from OpenStreetMap. Use of a real, harmonized agricultural dataset is largely absent. Our study addresses these gaps using the real-world EuroCrops SI-2021 dataset, additionally providing an up-to-date storage-size comparison of the four formats (RQ1) as a baseline.

3. Methodology

The main goal of this experimental study was to investigate the impact of GIS file formats and processing tools on vector spatial data loading, write speed, storage size, and query speed.

3.1. The dataset

This study is based on the Slovenian Crops data, a subset of EuroCrops, the largest harmonized open crop dataset across the European Union [6, 7]. The used Slovenian dataset covers the entire country and stores data for three years (2019, 2020, and 2021), with the latest year's dataset being harmonized and thus used in this study. For the selected year 2021, the dataset includes 828,161 agricultural parcels covering crop land use across Slovenia. The parcels were derived from the national GERK (Graphic Unit of Agricultural Land Use) registry, which is maintained by the Slovenian Ministry of Agriculture and is harmonized to the Hierarchical Crop and Agriculture Taxonomy (HCAT) within the EuroCrops framework [14]. Each parcel in the dataset has twelve attributes: (`ID`, `GERK_PID`), the Slovenian crop-use code (`SIFRA_KMRS`), area in m² (`AREA`), crop names in Slovenian, Latin/English, and translated form (`RASTLINA`, `CROP_LAT_E`, `EC_trans_n`), visualisation colour (`COLOR`), NUTS3 region (`EC_NUTS3`), harmonised HCAT name and code (`EC_hcat_n`, `EC_hcat_c`), and polygon geometry (`geometry`). Minor preprocessing was conducted to remove any lines with empty values ($N = 7$). The finalized dataset, therefore, included 828,154 polygons, of which 828,066 were classified as polygons and 88 as multipolygons. Based on the dataset's polygon coordinates, the average polygon is bounded by 9.91 coordinates ($Std = 4.18$), with the simplest polygon consisting of 4 coordinates and the most complex of 640. Altogether, 8,206,431 coordinates are used to represent all the polygons in the dataset. The filesize of the initial dataset, available in the geoparquet format, was 97.64 MB. The exploration of the dataset showed that the most common HCAT land cover categories are pasture meadows with grassland grass (57.7%), grain maize (14.6%), winter common soft wheat (5.3%), winter barley (4.4%), and clover (3.3%). Overall, 99 (of the available 150) different classifications were observed in the dataset.

3.1.1. Analyzed formats and geospatial data processing tools

Four formats were used in the benchmark comparison: GeoParquet, GeoPackage, GeoJSON, and CSV. They cover main categories of geospatial storage in common use, a columnar binary format (GeoParquet), a single-file spatial database (GeoPackage), a text-based interchange format (GeoJSON), and a flat tabular format (CSV/WKT). Performance of the formats was evaluated in the scope of two Python libraries, GeoPandas and DuckDB. They represent two different processing paradigms, with GeoPandas utilizing an established in-memory dataframe approach, and DuckDB utilizing an embedded analytical (columnar) engine.

3.2. Experiment

The benchmark was implemented in Python (version 3.12.0), using GeoPandas 1.1.3 (with pyogrio [0.12.1]) and DuckDB 1.5.2. The same dataset, previously described in section 3.1, was used for all four observed file formats (GeoParquet, GeoPackage, GeoJSON, and CSV) and two geospatial data processing tools, as presented in section 3.1.1. The benchmark was conducted via four independent

scenarios, as presented in Figure 1. *Scenario 1* covers reading. In it, the full file from disk is loaded into the processing tool's native in-memory representation. *Scenario 2* measures file writing. It includes serialization of the dataset already in the memory to disk in the target format. *Scenario 3* covers incremental updates via append. The source is loaded once, and each measured iteration concatenates a batch of n_a new records to the dataset and rewrites the entire file, for batch sizes $n_a \in \{100, 1000, 10000\}$. *Scenario 4* is focused on query execution performance and executes a set of analytical operations end-to-end (read followed by compute). The separation of loading from computation follows prior work, which likewise separated preparation (loading time) from query time [11].

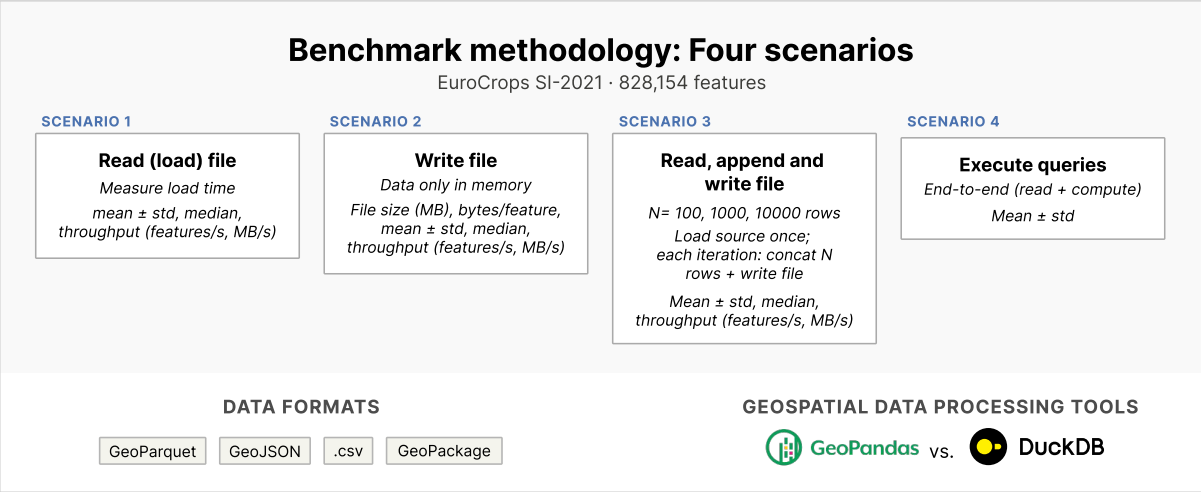


Figure 1: Experiment design: Four scenarios comparing four data formats and two geospatial data processing tools

All experiments were run on a single machine, whose system properties are presented in Table 1.

Table 1
System Properties.

CPU name	Intel Core i5-8265U
CPU clock speed	1.6-3.9 GHz
Cores (threads/core)	4 (2)
CPU architecture	Intel Whiskey Lake
Sockets	1
L1 data cache	32 KB per core
L1 instruction cache	32 KB per core
L2 unified cache size	256 KB per core
L3 unified cache size	6 MB shared
RAM capacity	16GB
Page Size	4 KB
Operating System	Windows 11 Pro, version 25H2

3.3. Metrics

Across all four scenarios, a common set of metrics is reported. The metrics were chosen to capture three aspects of format and tool performance: i) the time cost of individual operations, ii) throughput normalized by dataset size, so that formats with different on-disk footprints can be compared on equal terms, and iii) storage efficiency. Both timing and throughput were reported because a format can be fast in wall-clock time (measured with `perf_counter()`) while inefficient in bytes processed per second, or vice versa. Relying on either metric alone can therefore give a misleading picture of relative performance. The baseline for the analysis was a dataset with $N = 828{,}154$ rows. Each operation was

executed using five warmup (W) runs, whose data were discarded, followed by ten timed iterations (R) measured in wall-clock time. Reporting repeated runs after a warm-up phase follows the established practice in data-loading benchmarks. Wachs and Zacharatou [9] reported the average of ten executions, and Pandey et al. [11] submitted batches of queries to obtain stable timings.

Timing was reported via *mean* execution time together with *standard deviation* (*Std*) and *median* times. Mean values enable direct comparison, *Std* captures run-to-run predictability, and the median is reported to make any outlier-induced skew visible.

Throughput enables comparison independent of absolute runtime. *Features per second* (N/t) metric was used to represent how many geometries were processed per unit of time. As the dataset size (N) was fixed, this was the primary comparison measure. *Megabytes (MB) per second* S/t, where S is the file size in MB, indicates how many bytes are processed per unit time. A compact format may achieve a high features-per-second rate while moving comparatively few MBs, because it stores the same features in fewer bytes. Expressing throughput per feature mirrors prior work, which reports loading throughput as features processed per ms [9], though features/s were used in this work for readability.

Append throughput was used for the append scenario (scenario 3), where a randomly selected set of rows from the original dataset was concatenated to the end of the file. *Appended features per second* were reported as n_a/t, where $n_a \in \{100, 1000, 10000\}$ is the number of inserted rows. This is different from the total features/s rate (N/t). The `appended feature/s` measures the effective rate at which new records are absorbed, while the `total feature/s` metric reflects the cost of rewriting the entire file. The gap between the two quantifies the fixed overhead of a full-file rewrite relative to the size of the append.

Storage was reported via *file size* metric, which represents the on-disk size S of the serialised dataset, and *bytes per feature*, S_{bytes}/N, which normalizes file size by the number of geometries to provide insight into the storage efficiency of the format that is independent of dataset. Reporting on-disk size across formats is standard in format comparisons [9, 3].

Tool comparison included the *absolute times* with included *Std* over the ten runs as well as added *relative difference* in mean execution times $\Delta = (t_{\text{DuckDB}} - t_{\text{GeoPandas}})/t_{\text{GeoPandas}} \times 100\%$, using GeoPandas as the baseline. For this metric, negative values indicate that DuckDB is faster. Reporting both the absolute times and the relative difference makes the practical significance of the observed gaps visible without requiring the reader to recompute it from the raw timings.

4. Results

4.1. Storage size comparison

Storage size comparison was conducted by single-time measurement of the on-disk file size in MB on the first benchmark run after the dataset was written in each format and by each tool. The results, presented in Table 2, show that the GeoParquet format outperformed other formats by saving the dataset into three- (for CSV) to almost seven-times (for GeoJSON) smaller files, achieving the lowest byte per feature size of 123.6 (122.63 for DuckDB). The difference in file size between tools was none to minimal, the biggest difference of 10.95MB (1.6% difference in file size) was observed for GeoJSON format, this time with GeoPandas performing favorably. For GeoPackage, no difference was observed, while for GeoParquet and CSV, the difference of 0.74MB and 0.58MB was observed, with GeoPandas performing favorably in both cases.

4.2. Read speed comparison

Read speed was measured separately with DuckDB and GeoPandas by repetitive file reading with the use of `'SELECT * FROM'` query for DuckDB and format-specific read functions where available, such as `read_parquet()` and `read_file()` in GeoPandas. Measures of central tendency and dispersion are reported in Table 3 and indicate that the format has a larger impact on reading performance than the tool choice. GeoParquet in GeoPandas enabled the fastest reading, with the 828,154 feature file

Format	Tool	File size (MB)	Bytes/ feature
GeoParquet	GeoPandas	**97.64**	**123.63**
	DuckDB	**96.90**	**122.63**
GeoPackage	GeoPandas	349.82	442.92
	DuckDB	349.82	442.92
GeoJSON	GeoPandas	666.11	843.40
	DuckDB	677.06	857.27
CSV (WKT)	GeoPandas	291.55	369.15
	DuckDB	290.97	368.42

Bold values indicate the smallest file size for each tool.

Table 2: Storage size comparison of EuroCrops SI-2021 (828,154 features) by format and tool.

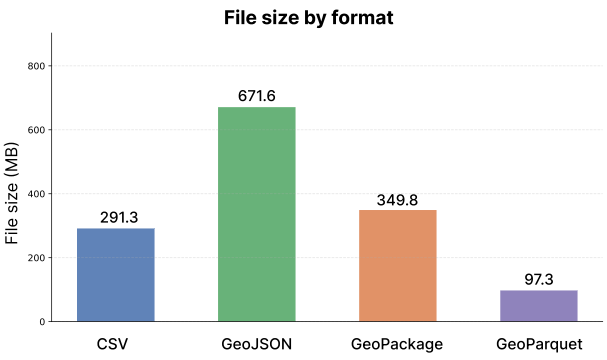


Figure 2: Storage size by formats.

being read on average in 1.5s ($Std = 0.23$). On the other hand, the same reading with DuckDB was on average 0.2s ($Std = 0.01$) slower. GeoJSON was consistently the slowest, with mean times being 32- to 44-times slower compared to GeoParquet. The tool with the best performance depends on the format. For the text formats, DuckDB enabled 7.77s (64%) better mean performance for CSV and 11.54s (17%) faster mean time for GeoJSON. GeoPackage enabled the highest mean MB/s reading (200), which was, however, not reflected in the highest features/s, as it was moving more bytes per second due to a much larger file size (350MB compared to GeoParquet's 97MB). GeoParquet had lower MB/s (57–65) yet the highest features/s, as its compression meant it read far fewer bytes to load the same features. GeoParquet also reached the highest stability in reading, as DuckDB reading GeoParquet has a near-zero deviation ($Std = 0.01$), which was the most consistent result in the set.

Table 3
Read (load-into-memory) performance, EuroCrops SI-2021, by tool and format.

Format	Tool	Mean (Std) (s)	Median (s)	Features/s	Mean MB/s
GeoParquet	GeoPandas	**1.5 (0.23)**	**1.35**	**552798.81**	65.18
	DuckDB	1.7 (0.01)	1.69	487626.86	57.49
GeoPackage	GeoPandas	1.75 (0.18)	1.67	474010.65	**200.22**
	DuckDB	1.75 (0.37)	1.54	473893.68	200.18
GeoJSON	GeoPandas	66.46 (1.18)	66.73	12461.40	10.02
	DuckDB	54.92 (1.91)	54.66	15080.52	12.13
CSV (WKT)	GeoPandas	12.11 (0.52)	12.20	68401.30	24.10
	DuckDB	4.34 (0.26)	4.26	190609.70	67.15

4.3. Write speed comparison

The speed of writing the dataset into a file was measured by timing the write operation for each format across multiple runs. Format-specific write functions were used in GeoPandas (`d.to_parquet()` and `d.to_file()`), while for DuckDB the `'COPY ... TO'` SQL statement was used. The file size was captured from disk after the first measurement write and stayed constant across runs. Results are presented in Table 4 and again indicate the best performance for GeoParquet, which was the fastest format in the writing comparison by a wide margin, reaching 183 MB/s writing speed. Comparison of the geospatial data processing tools shows that file writing was faster with DuckDB compared to GeoPandas for all observed formats, between 15% (GeoJSON) and 87% (GeoParquet). GeoJSON was again the last in comparison for write time and both throughput metrics. The most stable writing

results (with the least variance) were again observed with GeoParquet ($Std = 0.02$) used in DuckDB.

Table 4
Write performance, EuroCrops SI-2021.

Format	Tool	Mean (Std) (s)	Median (s)	Features/s	MB/s
GeoParquet	GeoPandas	4.04 (0.57)	3.95	204978.32	24.17
	DuckDB	**0.53 (0.02)**	0.53	1564373.76	**183.03**
GeoPackage	GeoPandas	18.40 (0.79)	18.14	45019.08	19.02
	DuckDB	5.15 (0.66)	4.77	160904.88	67.97
GeoJSON	GeoPandas	54.36 (0.96)	54.33	15234.96	12.25
	DuckDB	46.46 (1.41)	45.85	17826.92	14.57
CSV (WKT)	GeoPandas	14.87 (0.78)	14.82	55695.76	19.61
	DuckDB	2.71 (0.27)	2.61	305828.00	107.45

Bold values indicate the fastest write time and highest MB/s.

4.4. Append speed comparison

The append benchmark scenario simulated a real-world scenario where new rows were added to an existing EuroCrops dataset. The source file was read once upfront (not timed). For each append size (100, 1,000 and 10,000 rows) N randomly sampled rows from the source were used as the "new" data. Each timed run included two steps i) Appended rows were concatenated to existing dataset into a new GeoDataFrame, and ii) merged results were written to disk (same as in write scenario in section 4.3). The benchmark, therefore, measures the cost of the full read-modify-rewrite pattern, since formats like GeoParquet don't natively support in-place appending, the entire file must be rewritten. File size was captured after the first write, and the presented warmup-then-measure pattern was applied. The results presented in Table 5 and Figure 3 indicate that the append time remains consistent regardless of append size (Figure 3 (left), e.g., for CSV with DuckDB, the mean times are 2.66s, 2.65s, and 2.84s for 100, 1000, and 10000 rows, respectively) due to the whole dataset being re-serialized each time. The results indicate none of the formats support a true incremental append, so each operation re-serializes the entire 828-thousand row dataset regardless of how little is added. As writing time is constant, the time cost per row appended drops with the batch size. For DuckDB GeoParquet, throughput on just the appended rows rises from 181 features/s at batch 100 to 1,759 at batch 1,000 to 13,439 at batch 10,000 (see Figure 3 (right)). Performance ranking in this scenario remained unchanged from the previously observed write scenario. DuckDB was faster for every data format, and GeoParquet was the fastest of the observed formats, followed by CSV, GeoPackage, and lastly GeoJSON. Additionally, the combination of GeoJSON with DuckDB was the most variable, with a standard deviation reaching up to 6.18 on the mean time of 49.42s.

Table 5: Append-write performance, EuroCrops SI-2021, by format, tool, and append size.

Format	Tool	Append (rows)	Mean (Std) (s)	Median (s)	Features/s	Append feat/s	MB/s
GeoParquet	GeoPandas	100	4.57 (0.29)	4.65	181043	22	20.60
		1000	4.08 (0.44)	4.21	203231	245	23.13
		10000	4.32 (0.27)	4.28	194060	2315	22.13
	DuckDB	100	**0.55 (0.02)**	0.55	**1497891**	181	**175.26**
		1000	0.57 (0.04)	0.56	1458496	1759	170.69
		10000	0.74 (0.12)	0.74	1126405	13439	132.14

Continued on next page

Table 5: Append-write performance (continued).

Format	Tool	Append (rows)	Mean (Std) (s)	Median (s)	Features/s	Append feat/s	MB/s
GeoPackage	GeoPandas	100	18.61 (0.65)	18.65	44508	5	18.80
		1000	18.32 (1.03)	18.12	45253	55	19.12
		10000	18.45 (0.93)	18.32	45432	542	19.18
	DuckDB	100	5.05 (0.60)	4.88	164010	20	69.29
		1000	5.20 (0.66)	5.08	159501	192	67.38
		10000	4.94 (0.27)	4.90	169521	2023	71.57
GeoJSON	GeoPandas	100	54.43 (1.03)	54.52	15216	2	12.24
		1000	54.69 (0.82)	54.61	15160	18	12.19
		10000	55.19 (0.87)	54.96	15186	181	12.21
	DuckDB	100	48.51 (5.63)	47.02	17075	2	13.96
		1000	49.42 (6.18)	47.30	16776	20	13.72
		10000	48.89 (5.45)	47.18	17144	205	14.02
CSV (WKT)	GeoPandas	100	15.03 (0.74)	14.94	55121	7	19.41
		1000	15.42 (0.79)	15.78	53782	65	18.93
		10000	15.09 (0.76)	15.07	55560	663	19.56
	DuckDB	100	2.66 (0.08)	2.62	311321	38	109.38
		1000	2.65 (0.14)	2.67	312874	377	109.93
		10000	2.84 (0.31)	2.70	295375	3524	103.78

Bold values indicate the fastest append-write time, highest features/s and highest MB/s. Features/s is total throughput (all rows rewritten); Append feat/s counts only appended rows.

4.5. Query performance comparison

Query performance was evaluated with six representative queries selected with a goal to cover a range of analytical workloads: i) an attribute filter (parcels with crop type ”potatoes”), ii) a group-by aggregation (total area per crop category), iii) a per-region dominant crop calculation requiring a windowed aggregation, and iv) a geometric area calculation derived directly from the geometry column, and v) a spatial join query, finding all parcels within region SI035 that spatially intersect the 5 largest parcels in that region, and vi) a geometry dissolve query, merging parcel geometries within a single NUTS3 region. The queries were built with increasing computational complexity, from simple input-output to more analytically complex ones. Time is measured end-to-end, meaning the file reading and querying are both included. The resulting descriptive statistics are reported in Table 6. The comparison reveals that DuckDB was faster than GeoPandas in 23/24 format-query combinations. The gap narrowed for dissolve by region (Q6) regardless of format, which is consistent with the interpretation that geometry-heavy computation, not I/O, becomes the bottleneck. The one exception was GeoPackage, where DuckDB was marginally slower than GeoPandas on dissolve (+4.5%), the only case where the tool ranking reversed. The magnitude of DuckDB’s advantage depends on the format: it is largest for GeoParquet, where aggregation queries benefit from columnar pushdown (−98.5% for group-by, −96.7% for the region aggregation), and comparatively modest for GeoPackage and GeoJSON, where the underlying I/O and geometry-decoding costs already dominate for both tools. The largest gains can be observed with GeoParquet, for pure tabular aggregation, where group by category (Q2, 99%) and group by region (Q3, 97%) were used. These queries specifically show what a vectorized analytical engine of DuckDB is built for. Spatial join on neighboring plots (Q5) was 86% faster, which is similar to area calculation (Q4, 75%) and filter by crop (Q1, 70%) queries, though the presence of geometric operations lowered the difference between the tools compared to prior pure aggregations. For GeoParquet, the narrowest gap (19%) was observed for the dissolve by region query (Q6), the most geometry-heavy operation in the set. Across the other formats, the tool gap could narrow down to 16.8% for GeoJSON on Q3 and the reversed +4.5% for GeoPackage on Q6, confirming that once a workload is dominated by parsing or geometry computation rather than I/O and aggregation, the choice of engine matters far less. Notably, dissolve is also the one query where GeoParquet’s format advantage largely disappears, with GeoPandas, GeoPackage (4.93 s) slightly outperformed GeoParquet (5.18 s), and even with DuckDB the

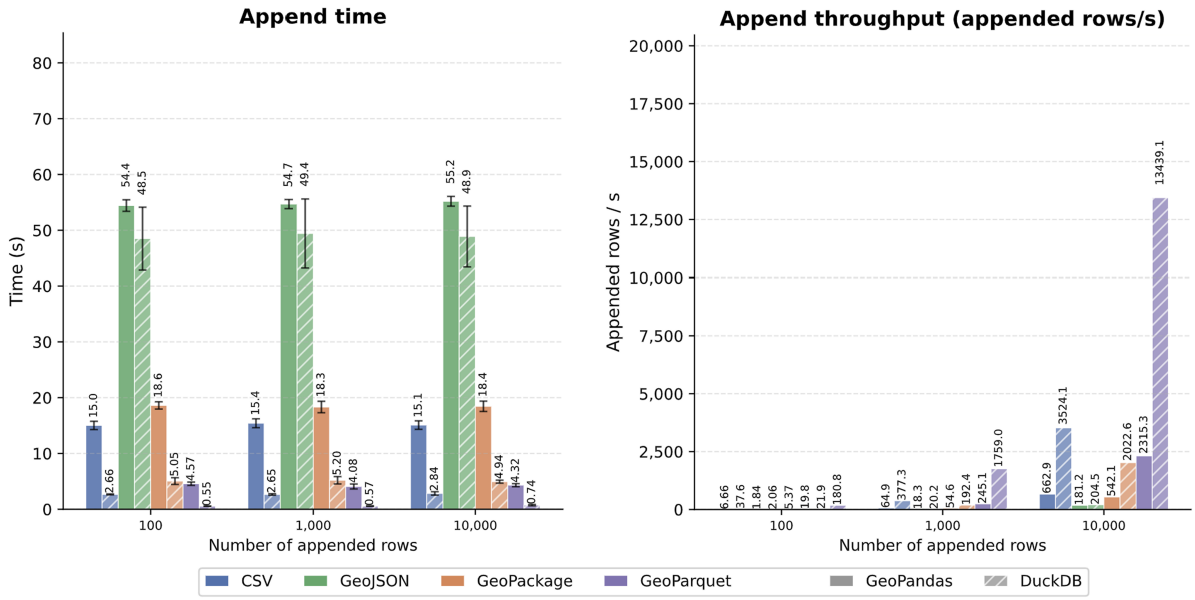


Figure 3: Comparison of data formats and tools on append time and append throughput (Scenario 3)

margin over GeoPackage was modest.

Table 6: GeoPandas vs. DuckDB query performance by format, EuroCrops SI-2021.

Query	Format	Mean GeoPandas (Std) (s)	Mean DuckDB (Std) (s)	Difference (%)
Q1: filter by crop	GeoParquet	1.65 (0.24)	0.49 (0.04)	−70.4
	GeoPackage	1.80 (0.25)	0.85 (0.25)	−53.0
	GeoJSON	73.90 (3.88)	55.61 (3.00)	−24.7
	CSV (WKT)	12.28 (0.50)	3.11 (0.46)	−74.7
Q2: group-by area by category	GeoParquet	1.51 (0.26)	0.02 (0.003)	−98.5
	GeoPackage	1.93 (0.24)	0.94 (0.05)	−51.2
	GeoJSON	79.27 (0.57)	64.80 (0.95)	−18.3
	CSV (WKT)	12.08 (0.54)	2.57 (0.37)	−78.7
Q3: dominant crop by region	GeoParquet	1.62 (0.30)	0.05 (0.002)	−96.7
	GeoPackage	1.80 (0.06)	0.94 (0.01)	−47.5
	GeoJSON	78.54 (1.52)	65.36 (1.02)	−16.8
	CSV (WKT)	12.38 (0.50)	2.58 (0.42)	−79.2
Q4: calculate area	GeoParquet	1.55 (0.20)	0.38 (0.03)	−75.4
	GeoPackage	1.99 (0.24)	0.87 (0.28)	−56.1
	GeoJSON	69.36 (1.81)	51.73 (1.46)	−25.4
	CSV (WKT)	12.24 (0.68)	6.26 (0.64)	−48.9
Q5: spatial join (neighbors)	GeoParquet	1.65 (0.29)	0.23 (0.004)	−86.1
	GeoPackage	1.72 (0.04)	0.85 (0.24)	−50.3
	GeoJSON	76.17 (3.41)	53.37 (3.38)	−29.9
	CSV (WKT)	12.17 (0.40)	2.77 (0.45)	−77.2
Q6: dissolve by region	GeoParquet	5.18 (0.51)	4.16 (0.20)	−19.6
	GeoPackage	4.93 (0.45)	5.15 (0.56)	+4.5
	GeoJSON	82.65 (1.65)	57.31 (1.29)	−30.7
	CSV (WKT)	15.46 (0.96)	6.94 (0.54)	−55.1

Difference is calculated as (DuckDB − GeoPandas)/GeoPandas; negative value means that DuckDB is faster.

5. Discussion

For vector data at the scale analyzed in this experiment (800k features, 12 attributes, mostly polygon geometry), the format in which the data is stored had a larger impact compared to which tool was used to process them. Adding to this, DuckDB generally performed better in the observed scenarios. GeoParquet stands out as the strongest format across every measured operation, compared to other observed formats. It was most compacted on disk (RQ1), and enabled the fastest reading and writing (RQ2 and RQ3). For writing, it was several times faster than the next-best format, CSV (see Table 4). GeoParquet's small file size confirmed it is an efficient format for columnar serialization. GeoParquet file written through DuckDB exhibited the lowest variability between runs of any configuration, so where predictable write latency matters in a pipeline, GeoParquet via DuckDB should be considered as a favorable setting. On the other end, GeoJSON was consistently the slowest and largest format and has proved to be impractical as a working format for data at this scale, and would be best reserved for interchange rather than computation.

An interesting pattern of inverted format ranking can be observed between reading and writing. On reads, GeoPackage was among the fastest formats (surpassed in mean and median time only by GeoParquet) while CSV was slow, owing to the cost of parsing WKT geometries. On file writing, the order was reversed; CSV was cheap to write, while GeoPackage was comparatively costly (faster only compared to GeoJSON). Most likely because constructing the GeoPackage container and its associated structures added write-time overhead that is not included in a flat CSV.

The append scenario (Scenario 3) revealed a behavior common to all formats. Comparing Tables 5 and 4, append times were effectively equal to full-write times as none of the observed formats appended new rows incrementally, but instead re-serialized the entire dataset on every operation. Appending a single batch costs essentially the same as rewriting the file from scratch. Because the cost is fixed per operation, the per-appended-row efficiency scales directly with batch size. The practical implication for practitioners is clear: append batches should be made as large as possible, and pipelines should favor large batched writes over row-by-row appends.

Finally, comparing different geospatial data processing tools (RQ4), for read, query, and write operations, DuckDB was generally faster than GeoPandas. For GeoParquet, it reduced execution time for filtering, aggregation, full-file writing, and appending by between roughly 70% and 99% (Table 6). The advantage was not uniform across operation types. DuckDB's advantage was largest for input/output and aggregation-heavy work, and smallest for geometry-bound work. On the most geometry-heavy query (dissolve), GeoPandas and DuckDB achieved comparable performance (the smallest difference between tools, 19% for GeoParquet). When the spatial algorithm itself, rather than data movement or aggregation, is the bottleneck, the choice of engine matters far less. The key qualitative finding of the engine comparison is that DuckDB became comparable to GeoPandas where the workload becomes compute-bound on geometry rather than bound by input/output or aggregation.

These results extend prior findings on geospatial storage-format performance. Wachs and Zacharatou [9] reported that GeoParquet delivered the highest loading throughput among compared formats on synthetic vector data and a real-world OpenStreetMap extract for Denmark. Our measurements on the EuroCrops SI-2021 dataset confirmed this ranking and extended it to writing, appending, and query execution, indicating that the columnar advantage holds across the full data lifecycle, not just reads. The observed on-disk sizes (RQ1) are consistent with Saeedan and Eldawy [3], who reported substantial storage reductions for columnar spatial formats over row-oriented alternatives; on our dataset, GeoParquet was roughly three times smaller than CSV and seven times smaller than GeoJSON. Middleton and Di [8] found a comparable format ranking on raster land-cover data from Sentinel-2 imagery, which our study complements with vector-focused evidence at the scale of a national agricultural parcel registry. The strong performance of GeoParquet with DuckDB also aligns with practitioner observations by Holmes [13], here quantified across read, write, append, and query workloads rather than loading alone. This advantage has a limit, however: once cost shifts from I/O to the geometric computation itself, as with the dissolve query, the gains from both format and engine choice shrink sharply.

6. Conclusion

This manuscript presents a systematic benchmark of four geospatial formats, GeoParquet, GeoPackage, GeoJSON, and CSV (WKT) across two different geospatial data processing tools, GeoPandas and DuckDB, on the real-world agricultural EuroCrops SI-2021 dataset. The comparison covered storage size (RQ1), read (RQ2), write and incremental inserts (RQ3), and tool-based performance (RQ4). The results show that the format is the dominant factor in performance, while the choice of the geospatial data processing tool is secondary. GeoParquet emerged as the best choice in performance across all observed scenarios except the most geometry-heavy query, where format differences largely disappeared. It was the most compact in file size, the fastest to read, write, and append, as well as the most stable one. GeoJSON should be avoided for datasets of this size in both reading and writing. Comparing the geospatial data processing tools, DuckDB was generally faster, especially for filtering and aggregation, but became comparable in geometry-heavy computations. Across all formats, appends are rewrite-bound, so incremental-update pipelines should batch writes as large as possible. Overall, these findings offer concrete guidance for selecting a storage format and processing engine for large vector geospatial workloads and indicate that an analysis-ready columnar format, such as GeoParquet, yields the largest and most consistent gains.

6.1. Limitations

Several limitations should be considered. Read comparison differs between the geospatial data processing tools, as GeoPandas decodes geometries into Shapely objects on load [15], and DuckDB keeps them as WKB blobs and decodes them only when needed [16]. Faster DuckDB load times partly reflect less geometry work. Next, DuckDB was also evaluated as a Python library to allow a better comparison with GeoPandas, even though the database engine on its own could be benchmarked. The selected queries were built to be as representative as possible, but still did not include window queries, and the only spatial join was a within-region self-join rather than a cross-dataset join. Most queries returned a moderate fraction of the data; a more selective query could better expose predicate-pushdown and row-group-skipping advantages of columnar formats (GeoParquet). The experiment was run on a single machine with a single hardware and operating system configuration, reading files from local storage with a warm cache. Cache effects were not systematically controlled, i.e., a cold-cache condition was not tested, and the gap between formats may narrow or widen once caching is removed from the measurement. The absolute timings are therefore specific to this setup, and the results may not transfer directly to a substantially different hardware, storage media, or cloud object storage.

6.2. Future work

Two directions for future research are indicated by this study. First, this study used only a single dataset, with predominantly simple polygons in its geometry. Future work should examine datasets with greater variation in geometric content. Because the cost of decoding and serializing grows with their complexity, such datasets would test the robustness of the compared data formats and may reveal, if GeoParquet's compactness and DuckDB's WKB handling keep their advantage as geometries become more (in case of multipolygons) or less (in case of points) complex. Second, queries selected in this study cover a limited range of geometry-intensive operations. As the results indicated that DuckDB's advantage was lower in computationally complex queries (dissolve by region), additional benchmarks would be helpful to explore at which point (if at all), GeoPandas could outperform DuckDB. Identifying such a crossover would clarify the boundary between workloads where an analytical engine, such as DuckDB, dominates and those where an in-memory geometry library (GeoPandas) remains competitive. Future work should also replicate the scenarios under a wider range of hardware and storage conditions (e.g., HDD, cloud object storage) and with cold-cache measurements to isolate pure I/O cost.

Acknowledgments

The authors acknowledge the financial support from the Slovenian Research Agency (Research Core Funding No. P2-0057) and the University of Maribor (Funding for project GeoData Lakehouse - platform for OLTP and OLAP for geospatial data, RSF ZRD UM 2023–2027).

Declaration on Generative AI

The authors declare that AI tools were used solely for translation purposes and to improve language clarity. All content, ideas, and conclusions are the authors' own, and the translated material has been reviewed for accuracy.

References

[1] Copernicus Data Space Ecosystem, Copernicus Data Space Ecosystem: Annual Report 2024, Technical Report, European Space Agency, 2024. Accessed: 2026-06-18.

[2] National Aeronautics and Space Administration, 2024 program highlights, NASA Earthdata, 2024. URL: https://www.earthdata.nasa.gov/about/program-highlights/2024.

[3] M. Saeedan, A. Eldawy, Spatial parquet: a column file format for geospatial data lakes, in: Proceedings of the 30th International Conference on Advances in Geographic Information Systems, SIGSPATIAL '22, ACM, New York, NY, USA, 2022. doi:10.1145/3557915.3561038.

[4] Open Geospatial Consortium, GeoParquet: Geospatial Data in Apache Parquet, https://geoparquet.org/, 2026. Accessed: 2026-05-30.

[5] OpenStreetMap contributors, Openstreetmap statistics, https://wiki.openstreetmap.org/wiki/Stats, 2024. Accessed: 2026-06-18.

[6] M. Schneider, T. Schelte, F. Schmitz, M. Körner, Eurocrops: The largest harmonized open crop dataset across the european union, Scientific Data 10 (2023) 612. doi:10.1038/s41597-023-02517-0.

[7] M. Schneider, M. Körner, Eurocrops, 2022. URL: https://doi.org/10.5281/zenodo.7476474. doi:10.5281/zenodo.7476474.

[8] G. Middleton, L. Di, Accelerating agro-geoinformatics ml with geoparquet, 2025, pp. 1–5. doi:10.1109/Agro-Geoinformatics66479.2025.11136321.

[9] A. Wachs, E. T. Zacharatou, Analysis of geospatial data loading, in: Proceedings of the Tenth International Workshop on Testing Database Systems, DBTest '24, ACM, New York, NY, USA, 2024, p. 36–42. doi:10.1145/3662165.3662761.

[10] Open Geospatial Consortium, Testbed 20 High-Performance Geospatial Computing Optimized Formats Report, OGC Engineering Report OGC 24-044, Open Geospatial Consortium, 2024. URL: https://docs.ogc.org/per/24-044.html.

[11] V. Pandey, A. Kipf, T. Neumann, A. Kemper, How good are modern spatial analytics systems?, Proceedings of the VLDB Endowment 11 (2018) 1661–1673.

[12] S. Kim, Y. S. Kanwar, GeoYCSB: A benchmark framework for the performance and scalability evaluation of NoSQL databases for geospatial workloads, in: 2019 IEEE International Conference on Big Data (Big Data), IEEE, 2019, pp. 3666–3675.

[13] C. Holmes, Performance explorations of GeoParquet (and DuckDB), 2023. URL: https://cloudnativegeo.org/blog/2023/08/performance-explorations-of-geoparquet-and-duckdb/.

[14] Ministry of Agriculture, Forestry and Food of the Republic of Slovenia, Graphic units of agricultural holdings, register of agricultural holdings, https://rkg.gov.si/GERK/, 2021. Accessed: 2026-06-15.

[15] GeoPandas developers, GeoPandas documentation: Data structures, https://geopandas.org/en/stable/docs/user_guide/data_structures.html, 2024. Accessed: 2026-06-18.

[16] DuckDB, Spatial extension, https://duckdb.org/docs/stable/core_extensions/spatial/overview, 2024. Accessed: 2026-06-18.