

# gharc: A stream-and-filter tool for the GitHub Archive on consumer hardware

Arav Panwar<sup>1</sup>

<sup>1</sup> Independent Researcher

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

## Software

- [Review](#)
- [Repository](#)
- [Archive](#)

Editor: [Open Journals](#)

## Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

## License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

## Summary

gharc is a command-line tool and Python library that filters the GitHub Archive (GHArchive) dataset on consumer hardware. Rather than downloading hours of compressed events to local disk before processing, gharc streams each archive file through memory, decompresses it on the fly, applies user-specified repository and event-type filters, and writes only the matching events to Parquet or JSONL. Peak storage stays bounded by a single in-flight download (about 150 MB) regardless of the time range processed. The intended audience is software-engineering researchers, students, and small teams who do not have access to institutional warehouses or commercial cloud quotas but who need event-level GitHub data at multi-year time scales.

## Statement of need

The GitHub Archive ([Grigorik, 2012](#)) is one of the most widely used data sources in mining software repositories (MSR) research, capturing nearly every public event on GitHub since February 2011. The dataset is published as hourly gzipped JSONL files. Each hour in 2024 averages roughly 90 MB compressed and several hundred megabytes uncompressed; the full archive exceeds several petabytes uncompressed.

This scale poses a practical barrier to anyone without institutional infrastructure. Three traditional approaches each exclude part of the research community:

- Bulk local downloads.** Storing even a single year of GHArchive uncompressed exceeds the disk budget of most laptops. Filtering after the fact still requires the full download to be present.
- Cloud warehouses.** GHArchive is mirrored on Google BigQuery and Snowflake ([GH Archive contributors, 2024](#)). Both are excellent for query workloads, but they require a billing account, and exploratory analyses can quickly exceed free quotas.
- Hosted research infrastructures.** Tools such as GHTorrent ([Gousios, 2013](#)), Boa ([Dyer et al., 2013](#)), and World of Code ([Ma et al., 2019](#)) have provided shared access to GitHub-derived data, but each carries its own constraints. GHTorrent's project domain has been allowed to expire as of 2026, Boa's hosted endpoint is currently unreachable, and World of Code requires registration and remote access to dedicated servers.

gharc fills the laptop-friendly local-first niche. By streaming each hour's compressed archive through a small in-memory buffer and discarding it immediately after filtering, the tool keeps peak storage bounded while preserving the ability to operate on multi-year time ranges with no warehousing dependency. This is the configuration most useful to independent researchers, students, and small teams.

38 **Architecture**

gharc is a stream-and-filter pipeline. The user supplies a date range, an optional list of repositories, and an optional list of event types; gharc constructs the GHArchive URL for each hour in the range and dispatches them to a worker thread pool.

Each worker downloads the hour's gzipped JSONL file to a temporary location with HTTP range-resume support, then iterates over the file line by line. Lines are first filtered by a fast byte-level token check before any JSON parsing, so gharc skips the majority of irrelevant events without paying parser cost. Matching events are JSON-parsed, validated against the full filter, and yielded to the main thread.

The main thread receives matching events from completed futures and writes them to disk through a `DataWriter` that wraps a long-lived `pyarrow.parquet.ParquetWriter` (Apache Arrow contributors, 2024) for true streaming append. Nested fields (`actor`, `repo`, `payload`, `org`) are JSON-stringified before write so the Parquet schema is stable across heterogeneous event types within the same output file. Each worker maintains its own requests session (Reitz, 2011) in thread-local storage so HTTP connection pooling persists across the hours that worker processes.



**Figure 1: Stream-and-filter architecture.**

## 54 Performance

55 We measured gharc on a Windows 11 laptop (12 logical cores, 15 GB RAM) over a typical  
56 residential connection.

A six-hour window of GHArchive (2024-01-01 00:00 to 06:00 UTC), filtered to apache/spark, completed in 76 seconds with a single worker and 58 seconds with four workers. Both runs recovered the same 14 events, so concurrency does not affect output correctness. Peak resident set size stayed under 110 MB in both configurations. The limiting factor on residential links is HTTPS download throughput rather than CPU; additional workers beyond a small number add little once the connection saturates.

The same six-hour window comprises approximately 1.2 GB of compressed source data on the GHArchive side; the filtered Parquet output for apache/spark is 53 KB. That ratio of roughly 22,000 to 1 quantifies the storage saving from streaming-and-filtering. At no point did peak local disk exceed the size of a single in-flight temporary file (about 150 MB).

67 Extrapolating from these numbers, a full January to June 2024 fetch (4,380 hours of source  
68 data at about 90 MB per hour) is approximately 395 GB streamed, which is achievable in  
69 three to four evening sessions on the same hardware. The reproducible benchmark scripts are  
70 included in the repository under benchmarks/.

71 **Case study: Apache Spark, January to June 2024**

The motivation for gharc came from a six-month analysis of Apache Spark contributor activity originally conducted as an undergraduate course mini-project (Panwar, 2025). That earlier study downloaded GHArchive month by month, ran a separate filter script over each month, and ultimately recovered 40,237 events from 2,384 unique contributors across the six months. Of those, nine “core” contributors accounted for 53 percent of total activity, and 57 percent of all events were code review (PullRequestReviewEvent and PullRequestReviewCommentEvent), consistent with prior characterizations of large Apache projects as review-heavy.

79 That month-by-month pipeline was prone to off-by-one errors in date ranges (notably a  
80 “seventh-month bleed” caused by an inclusive end bound), and it required around 100 GB  
81 of intermediate local disk during processing. gharc reproduces the same analysis on the  
82 same laptop with no intermediate disk pressure and a single command. A full re-fetch using  
83 gharc is currently in progress and will be reported in future work, alongside a comparative  
84 case study across six high-volume projects (microsoft/vscode, go-lang/go, facebook/react,  
85 kubernetes/kubernetes, apache/spark, rust-lang/rust) for the period 2019 to 2024.

## 86 Related work

87 Several tools have addressed the GHArchive analysis problem, each with different trade-offs.

88 GHTorrent (Gousios, 2013) historically served as the default GitHub-mining database for MSR  
89 research and offered both periodic data dumps and a SQL-queryable mirror. As of 2026 the  
90 project’s primary domain redirects to an unrelated commercial site, illustrating the maintenance  
91 fragility of long-running shared services.

92 PyDriller (Spadini et al., 2018) mines git repositories directly via the local git history of  
93 cloned projects. It operates at a different layer to gharc: PyDriller exposes commits, file  
94 diffs, and developer information from a checked-out repository, whereas gharc operates on the  
95 GitHub event stream (issues, pull requests, reviews, watch events, and so on). The two are  
96 complementary; an MSR study often needs both.

97 World of Code (Ma et al., 2019) provides shared access to a curated cross-reference of millions  
98 of git repositories. Researchers obtain accounts and submit jobs to dedicated servers. The  
99 infrastructure is excellent for very large cross-project queries but introduces an institutional  
100 dependency.

101 Boa (Dyer et al., 2013) offers a domain-specific language for ultra-large-scale repository queries  
102 against curated datasets, again served from a hosted endpoint. At the time of writing the  
103 public endpoint at `boa.cs.iastate.edu` is unreachable.

104 Cloud-warehouse mirrors of GHArchive on Google BigQuery and Snowflake (GH Archive  
105 contributors, 2024) are highly performant but require a billing account and quota management.

106 gharc occupies the local-first, laptop-friendly niche: no shared infrastructure, no billing account,  
107 no schema-bound DSL, just the original GHArchive files streamed and filtered on demand.

## 108 Software availability

109 The source code is hosted at `github.com/aravpanwar/gharc` and the v0.1.0 release is archived  
110 on Zenodo (DOI: [10.5281/zenodo.19814233](https://doi.org/10.5281/zenodo.19814233)). The concept DOI [10.5281/zenodo.19814232](https://doi.org/10.5281/zenodo.19814232)  
111 always resolves to the latest archived version.

## 112 Acknowledgements

113 The author thanks Ilya Grigorik and the GHArchive maintainers for the public dataset on which  
114 this tool depends, and the maintainers of the requests (Reitz, 2011), pandas (McKinney,  
115 2010), pyarrow (Apache Arrow contributors, 2024), tqdm, and orjson libraries that gharc  
116 builds on.

## 117 References

118 Apache Arrow contributors. (2024). *Apache arrow*. <https://arrow.apache.org/>

- 119 Dyer, R., Nguyen, H. A., Rajan, H., & Nguyen, T. N. (2013). Boa: A language and  
120 infrastructure for analyzing ultra-large-scale software repositories. *Proceedings of the 2013*  
121 *International Conference on Software Engineering*, 422–431.
- 122 GH Archive contributors. (2024). *GH Archive on Google BigQuery*. <https://www.gharchive.org/#bigquery>.  
123
- 124 Gousios, G. (2013). The GHTorrent dataset and tool suite. *Proceedings of the 10th Working*  
125 *Conference on Mining Software Repositories*, 233–236.
- 126 Grigorik, I. (2012). *GH Archive: A GitHub timeline archive*. <https://www.gharchive.org/>.
- 127 Ma, Y., Bogart, C., Amreen, S., Zaretzki, R., & Mockus, A. (2019). World of code: An  
128 infrastructure for mining the universe of open source VCS data. *2019 IEEE/ACM 16th*  
129 *International Conference on Mining Software Repositories*, 143–154.
- 130 McKinney, W. (2010). Data structures for statistical computing in Python. *Proceedings of*  
131 *the 9th Python in Science Conference*, 445, 51–56.
- 132 Panwar, A. (2025). *Apache Spark codebase evolution: A six-month GHArchive analysis*.  
133 [https://github.com/aravpanwar/Spark\\_Codebase\\_Evolution](https://github.com/aravpanwar/Spark_Codebase_Evolution).
- 134 Reitz, K. (2011). *Requests: HTTP for humans*. <https://requests.readthedocs.io>
- 135 Spadini, D., Aniche, M., & Bacchelli, A. (2018). PyDriller: Python framework for mining  
136 software repositories. *Proceedings of the 2018 26th ACM Joint Meeting on European*  
137 *Software Engineering Conference and Symposium on the Foundations of Software*  
138 *Engineering*, 908–911.