

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

Arav Panwar¹

¹ 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Published: unpublished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 for extracting filtered subsets of the GitHub Archive (GHArchive) (Grigorik, 2012) on a personal computer. GHArchive records nearly every public GitHub event since 2011 and publishes it as hourly gzipped JSON-lines files; a single year is hundreds of gigabytes compressed and the full archive is several petabytes uncompressed. Most studies need only a slice of this, a few repositories or a few event types over a window of months, but the archive offers no server-side filter, so a researcher must obtain whole hours and discard almost all of them. gharc performs that selection as a stream: it downloads each hour to a temporary file, decompresses and filters it on the fly, writes only the matching events to Parquet or JSONL, and deletes the temporary file before moving on. Peak local storage is therefore set by the number of downloads in flight, one temporary file per worker, rather than by the length of the window processed. The intended users are software-engineering researchers, students, and small teams who need event-level GitHub data at multi-year scales but do not have a cloud-warehouse account or institutional storage.

Statement of need

The GitHub Archive is one of the most widely used data sources in mining software repositories (MSR) research. Its scale makes even modest studies awkward without institutional infrastructure, and the three common ways to work with it each impose a cost that excludes part of the research community. Bulk local downloads require enough disk to hold the raw hours before any filtering, which for a multi-month study exceeds a typical laptop. Cloud-warehouse mirrors on BigQuery and Snowflake (GH Archive contributors, 2024) are fast but require a billing account. Shared research infrastructures remove the storage problem but introduce an access dependency: an account, a remote job queue, or a maintained service that may lapse.

gharc targets the case these options leave open: a researcher on a personal machine, with no cloud account and bounded disk, who needs a specific slice of the archive over a long time range. By filtering each hour during the stream and keeping only the result, it makes the working storage independent of the window length, so the same laptop that can process one day can process a year. The motivating study was a six-month analysis of Apache Spark contributor activity (Panwar, 2025) whose original pipeline stored each month before filtering and peaked near 100 GB of intermediate disk; gharc has since reproduced that analysis end to end with bounded disk, and the reproduction is published alongside the original study.

State of the field

GHArchive is the data source gharc consumes rather than a competitor. Among the tools that help researchers turn it, or related GitHub data, into analyzable subsets, the useful distinction

40 is the access model each assumes.

41 Cloud-warehouse mirrors of GHArchive on BigQuery and Snowflake ([GH Archive contributors,](#)
42 [2024](#)) offer SQL over the full dataset but require a cloud billing account. GHTorrent ([Gousios,](#)
43 [2013](#)) was for years the default GitHub-mining database, offering periodic dumps and a
44 queryable mirror; its data collection has not been actively maintained since around 2019, with
45 the most recent dumps from 2021, which illustrates how centrally hosted mirrors can lapse. Boa
46 ([Dyer et al., 2013](#)) provides a domain-specific language for ultra-large-scale repository queries
47 against curated datasets, accessed through a hosted service that requires account registration
48 and approval. World of Code ([Ma et al., 2019](#)) gives researchers a curated cross-reference
49 of millions of repositories through accounts and jobs submitted to dedicated servers. These
50 shared infrastructures are powerful for cross-project queries at a scale a laptop cannot reach,
51 but each adds an external dependency. PyDriller ([Spadini et al., 2018](#)) works at a different
52 layer, mining commits and diffs from cloned git repositories rather than the GitHub event
53 stream, and is complementary to gharc.

54 gharc occupies the remaining position: local-first, no account, no query service, and no
55 schema-bound language, just the original GHArchive files streamed and filtered on demand.
56 That position is a deliberate trade-off, described next.

57 Software design

58 gharc is built around one decision: decouple peak local storage from the time range by treating
59 each hour as transient. The user supplies a date range and optional filters on repository (an
60 exact name or an owner/* wildcard), owner, actor, and event type; gharc builds the GHArchive
61 URL for each hour and dispatches the hours to a worker pool. Each worker downloads an
62 hour, filters it line by line, and returns the matches; the main thread writes them out and the
63 temporary file is removed ([Figure 1](#)).

64 The central trade-off follows from that decision. Streaming and filtering re-pays download
65 bandwidth on every run, because nothing is cached locally for reuse, and it deliberately offers
66 no cross-time query capability. In exchange, peak disk stays bounded and the tool needs no
67 database, warehouse, or persistent local dataset. For the target setting, repeated reads of the
68 same window are rare and disk is the binding constraint, so the design favors bounded storage
69 over reuse, and leaves aggregation and joins to whatever the researcher already uses (pandas,
70 Polars, DuckDB, Spark) over the small filtered output.

71 Three further design choices address correctness and robustness rather than storage.

72 *Filtering before parsing.* Within each hour a byte-level token check rejects lines that cannot
73 match before any JSON parsing; only surviving lines are parsed and checked against the
74 structured filter. The token check is a superset test, so it may admit a false positive that the
75 structured filter then rejects, but it never discards a true match. On a selective filter this skips
76 the large majority of lines for the cost of a substring scan; on a wide or empty filter it saves
77 nothing and the run pays full parsing cost, which is the honest boundary of the optimization.

78 *Stable columnar output across heterogeneous events.* GitHub events share a top-level shape
79 but differ in practice: the org field appears only for organization-owned repositories, and
80 nested fields vary by event type. Writing Parquet incrementally over a stream turns this into a
81 correctness issue, because a schema inferred from the first batch will reject or silently drop later
82 batches whose columns differ. gharc pins an explicit top-level schema and JSON-stringifies
83 nested fields, so every appended batch conforms regardless of which event types or ownership
84 patterns occur in a given hour, and a run produces one file that reads back as a single
85 table. This was a design lesson: an earlier first-batch-inference approach corrupted output on
86 mixed-ownership runs.

87 *Failure semantics for unreliable connections.* Because the target setting is residential internet,
88 gharc separates a download that failed from an hour that genuinely had no matches. A

89 persistent download failure is reported with a non-zero exit rather than recorded as an empty
90 hour, so a flaky link cannot silently yield an incomplete dataset, while a genuinely absent
91 archive hour is treated as zero events. Completed hours are recorded in a sidecar checkpoint
92 fingerprinted by the run parameters, so an interrupted run resumes the remaining hours instead
93 of restarting. Resume is supported for JSONL output, since a closed Parquet file cannot be
94 appended to; the tool makes that asymmetry explicit and recommends JSONL with a later
95 conversion step for long runs.

96 Two scope boundaries shape how the tool should be used. It is bound by HTTPS download
97 throughput rather than local CPU, so adding workers beyond a few gives diminishing returns
98 once a residential connection saturates, and matching events for an hour are buffered before
99 writing, so working memory stays near 100 MB for selective filters but grows for very wide
100 ones. GHArchive also uses the GitHub Events API schema from January 2015 onward and an
101 older Timeline API schema before that; gharc does not normalize across that boundary, so
102 studies spanning 2011 to 2014 should expect some fields to be missing or shaped differently.

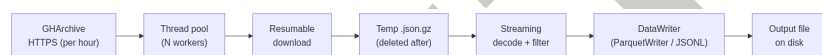


Figure 1: Stream-and-filter architecture.

103 Research impact statement

104 gharc was motivated by a prior six-month study of Apache Spark contributor activity (Panwar,
105 2025). The earlier pipeline downloaded GHArchive month by month, filtered each month with
106 a separate script, peaked near 100 GB of intermediate disk, and contained a date off-by-one
107 error in which an inclusive end bound pulled in an extra month. gharc removes both problems:
108 storage stays bounded and the date range is end-exclusive by construction.

109 That study has now been reproduced with gharc 0.1.4 pinned from PyPI, and the reproduction
110 is published in the original study's repository (Panwar, 2025). The full window (January to
111 June 2024, 4,368 hourly files, roughly 350 GB of compressed source) completed in 11.3 hours
112 on a consumer laptop over a residential connection, with peak temporary disk of 565 MB
113 against the original's roughly 100 GB, producing a 112 MB Parquet dataset. Every finding of
114 the original study reproduced, including the exact event count of its most active contributor.
115 The rerun also recovered 257 more events (0.6%) than the original, spread across all event
116 types, which is consistent with the original pipeline's download loop silently skipping hours
117 that failed to transfer; gharc reports failed hours and exits non-zero rather than recording
118 them as empty, so that class of silent data loss surfaces instead of being absorbed.

119 More broadly, the tool lowers the entry cost for event-level GitHub research to a laptop and
120 an internet connection, the configuration available to students and independent researchers
121 without cloud quotas. Reproducible benchmark scripts are included in the repository, the
122 package is distributed on PyPI with an automated test suite, and tagged releases are archived
123 on Zenodo, so analyses that depend on it can pin and cite a fixed version.

124 AI usage disclosure

125 The author wrote the implementation, the documentation, and this paper, and made all
126 software design decisions. Generative AI assistance (Claude Opus 4.8, Anthropic) was used
127 to write the automated test suite, to fix and harden bugs identified during development and
128 review, and to edit and polish the prose of the documentation and this paper. Correctness was
129 verified by the author through the automated test suite and live runs against GHArchive that
130 confirmed event counts, output integrity, and the reported benchmarks. The author takes full
131 responsibility for the software and the contents of this paper.

132 Software availability

133 The source code is hosted at github.com/aravpanwar/gharc under the MIT license. Tagged
134 releases are archived on Zenodo; the concept DOI [10.5281/zenodo.19814232](https://doi.org/10.5281/zenodo.19814232) always resolves
135 to the latest archived version.

136 Acknowledgements

137 The author thanks Ilya Grigorik and the GHArchive maintainers for the public dataset on which
138 this tool depends, and the maintainers of the requests ([Reitz, 2011](#)), pyarrow ([Apache Arrow](#)
139 [contributors, 2024](#)), tqdm, and orjson libraries that gharc builds on.

140 References

- 141 Apache Arrow contributors. (2024). *Apache arrow*. <https://arrow.apache.org/>
- 142 Dyer, R., Nguyen, H. A., Rajan, H., & Nguyen, T. N. (2013). Boa: A language and
143 infrastructure for analyzing ultra-large-scale software repositories. *Proceedings of the 2013*
144 *International Conference on Software Engineering*, 422–431. [https://doi.org/10.1109/](https://doi.org/10.1109/ICSE.2013.6606588)
145 [ICSE.2013.6606588](https://doi.org/10.1109/ICSE.2013.6606588)
- 146 GH Archive contributors. (2024). *GH Archive on Google BigQuery*. [https://www.gharchive.](https://www.gharchive.org/#bigquery)
147 [org/#bigquery](https://www.gharchive.org/#bigquery).
- 148 Gousios, G. (2013). The GHTorrent dataset and tool suite. *Proceedings of the 10th Working*
149 *Conference on Mining Software Repositories*, 233–236. [https://doi.org/10.1109/MSR.](https://doi.org/10.1109/MSR.2013.6624034)
150 [2013.6624034](https://doi.org/10.1109/MSR.2013.6624034)
- 151 Grigorik, I. (2012). *GH Archive: A GitHub timeline archive*. <https://www.gharchive.org/>.
- 152 Ma, Y., Bogart, C., Amreen, S., Zaretski, R., & Mockus, A. (2019). World of code: An
153 infrastructure for mining the universe of open source VCS data. *2019 IEEE/ACM 16th*
154 *International Conference on Mining Software Repositories*, 143–154. [https://doi.org/10.](https://doi.org/10.1109/MSR.2019.00031)
155 [1109/MSR.2019.00031](https://doi.org/10.1109/MSR.2019.00031)
- 156 Panwar, A. (2025). *Apache Spark codebase evolution: A six-month GHArchive analysis*.
157 https://github.com/aravpanwar/Spark_Codebase_Evolution.
- 158 Reitz, K. (2011). *Requests: HTTP for humans*. <https://requests.readthedocs.io>
- 159 Spadini, D., Aniche, M., & Bacchelli, A. (2018). PyDriller: Python framework for mining
160 software repositories. *Proceedings of the 2018 26th ACM Joint Meeting on European*
161 *Software Engineering Conference and Symposium on the Foundations of Software*
162 *Engineering*, 908–911. <https://doi.org/10.1145/3236024.3264598>