# grepq: A Rust application that quickly filters FASTQ files by matching sequences to a set of regular expressions

Nicholas D. Crosbie [1]¶

1 Melbourne Veterinary School, University of Melbourne, Parkville, Victoria, Australia ¶ Corresponding author

Summary

Regular expressions (regex) (Kleene, 1951) have been an important tool for finding patterns in biological codes for decades (Hodgman, 2000 and citations therein), and unlike fuzzy-finding approaches, do not result in approximate matches. The performance of regular expressions can be slow, however, especially when searching for matching patterns in large files. *grepq* is a Rust application that quickly filters FASTQ files by matching sequences to a set of regular expressions. *grepq* is designed with a focus on performance and scalability, is easy to install and easy to use, enabling users to quickly filter large FASTQ files, to enumerate named and unnamed variants, to update the order in which patterns are matched against sequences through built-in *tune* and *summarise* commands, and optionally, to output a SQLite file for further sequence analysis. *grepq* is open-source and available on *GitHub*, *Crates.io* and *bioconda*.

Statement of need

The ability to quickly filter FASTQ files by matching sequences to a set of regular expressions is an important task in bioinformatics, especially when working with large datasets. The importance and challenge of this task will only grow as sequencing technologies continue to advance and produce ever larger datasets (Katz et al., 2022). The use cases of *grepq* are diverse, and include pre-processing of FASTQ files before downstream analysis, quality control of sequencing data, and filtering out unwanted sequences. Where decisions need to be made quickly, such as in clinical settings (Bachurin et al., 2024), biosecurity (Valdivia-Granda, 2012), and wastewater-based epidemiology in support of public health measures (Choi et al., 2018; Merrett et al., 2024; Sims & Kasprzyk-Hordern, 2020; Xylogiannopoulos, 2021), the ability to quickly filter FASTQ files and enumerate named and unnamed variants by matching sequences to a set of regular expressions is attractive as it circumvents the need for more time-consuming bioinformatic workflows.

Regular expressions are a powerful tool for matching sequences, but they can be slow and inefficient when working with large datasets. Furthermore, general purpose tools like *grep* (Free Software Foundation, 2023) and *ripgrep* (A. Gallant, 2025) are not optimized for the specific task of filtering FASTQ files, and occassionally yield false positives as they scan the entire FASTQ record, including the sequence quality field. Tools such as *awk* (Aho et al., 1988) and *gawk* (Free Software Foundation, 2024) can be used to filter FASTQ files without yielding false positives, but they are significantly slower than *grepq* and can require the development of more complex scripts to achieve the same result.

Implementation

grepq obtains its performance and reliability, in part, by using the *seq_io* (Schlegel & Seyboldt, 2025) and *regex* (Gallant & others, 2025b) libraries. The *seq_io* library is a well-tested library for parsing FASTQ files, designed to be fast and efficient, and includes a module for parallel processing of FASTQ records through multi-threading. The *regex* library is designed to work with regular expressions and sets of regular expressions, and is known to be one of the fastest regular expression libraries currently available (Gallant & others, 2025a). The *regex* library supports Perl-like regular expressions without look-around or backreferences (documented at https://docs.rs/regex/1.*/regex/#syntax).

Feature set

- support for presence and absence (inverted) matching of a set of regular expressions
- IUPAC ambiguity code support (N, R, Y, etc.)
- support for gzip and zstd compression (reading and writing)
- JSON support for pattern file input and *tune* and *summarise* command output, allowing named regular expression sets and named regular expressions (pattern files can also be in plain text)
- the ability to:
  - set predicates to filter FASTQ records on the header field (= record ID line) using a regular expression, minimum sequence length, and minimum average quality score (supports Phred+33 and Phred+64)
  - output matched sequences in one of four formats (including FASTQ and FASTA)
  - tune the pattern file and enumerate named and unnamed variants with the *tune* and *summarise* commands: these commands will output a plain text or JSON file with the patterns sorted by their frequency of occurrence in the input FASTQ file or gzip-compressed FASTQ file (or a user-specified number of total matches). This can be useful for optimizing the pattern file for performance, for example by removing patterns that are rarely matched and reordering nucleotides within the variable regions of the patterns to improve matching efficiency
  - count and summarise the total number of records and the number of matching records (or records that don't match in the case of inverted matching) in the input FASTQ file
  - bucket matching sequences to separate files named after each regexName with the **−bucket** flag, in any of the four output formats

Other than when the **inverted** command is given, output to a SQLite database is supported with the **writeSQL** option. The SQLite database will contain a table called **fastq_data** with the following fields: the fastq record (header, sequence and quality fields), length of the sequence field (length), percent GC content (GC), percent GC content as an integer (GC_int), number of unique tetranucleotides in the sequence (nTN), number of unique canonical tetranucleotides in the sequence (nCTN), percent tetranucleotide frequency in the sequence (TNF), percent canonical tetranucleotide frequency in the sequence (CTNF), and a JSON array containing the matched regex patterns, the matches and their position(s) in the FASTQ sequence (variants). If the pattern file was given in JSON format and contained a non-null qualityEncoding field, then the average quality score for the sequence field (average_quality) will also be written. The **−num-tetranucleotides** option can be used to limit the number of tetranucleotides written to the TNF field of the fastq_data SQLite table, these being the most or equal most frequent tetranucleotides in the sequence field of the matched FASTQ records. A summary of the invoked query (pattern and data files) is written to a second table called **query**.

Performance

The performance of *grepq* was compared to that of *fqgrep*, *seqkit grep*, *ripgrep*, *grep*, *awk*, and *gawk* using the benchmarking tool *hyperfine*. The test conditions and results are shown in **Table 1**, **Table 2** and **Table 3** (see supplemental).

Testing and availability

grepq is open-source and available at *GitHub* (https://github.com/Rbfinch/grepq), *Crates.io* (https://crates.io/crates/grepq), and *bioconda* (https://anaconda.org/bioconda/grepq), and is distributed under the MIT license. It has been tested on macOS 15.0.1 (Apple M1 Max) and Linux Ubuntu 20.04.6 LTS (AMD EPYC 7763 64-Core Processor). For more information on the testing of *grepq*, see the *README.md* file in the *grepq* repository on *GitHub*.

Conclusion

The performance of *grepq* was compared to that of *fqgrep*, *seqkit grep*, *ripgrep*, *grep*, *awk*, and *gawk* using the benchmarking tool *hyperfine*. For an uncompressed FASTQ file 874MB in size, containing 869,034 records, *grepq* was significantly faster than the other tools tested, with a speedup of 1797 times relative to *grep*, 864 times relative to *awk*, and 19 times relative to *ripgrep*. For a larger uncompressed FASTQ file (104GB in size, and containing 139,700,067 records), *grepq* was 4.4 times faster than *ripgrep* and marginally slower or of equivalent speed to *ripgrep* where the same large file was gzip-compressed. When coupled with its exceptional runtime performance, *grepq*'s feature set makes it a powerful and flexible tool for filtering large FASTQ files.

Acknowledgements

Conflicts of interest

The author declares no conflicts of interest.

References

Aho, A. V., Kernighan, B. W., & Weinberger, P. J. (1988). *The AWK programming language*. https://www.cs.princeton.edu/~bwk/btl.mirror/

Bachurin, S. S., Yurushkin, M. V., Slynko, I. A., Kletskii, M. E., Burov, O. N., & Berezovskiy, D. P. (2024). Structural peculiarities of tandem repeats and their clinical significance. *Biochemical and Biophysical Research Communications*, *692*, 149349. https://doi.org/10.1016/j.bbrc.2023.149349

Choi, P. M., Tscharke, B. J., Donner, E., O'Brien, J. W., Grant, S. C., Kaserzon, S. L., Mackie, R., O'Malley, E., Crosbie, N. D., Thomas, K. V., & others. (2018). Wastewater-based epidemiology biomarkers: Past, present and future. *TrAC Trends in Analytical Chemistry*, *105*, 453–469. https://doi.org/10.1016/j.trac.2018.06.004

Free Software Foundation. (2023). *GNU grep 3.11*. Free Software Foundation. https://www.gnu.org/software/grep/manual/grep.html

Free Software Foundation. (2024). *GAWK: Effective AWK programming: A user's guide for GNU awk, for the 5.3.1*. Free Software Foundation. https://www.gnu.org/software/gawk/manual/gawk.html

Gallant, A. (2025). *Ripgrep: Recursively search the current directory for lines matching a pattern* (Version 14.1.1). https://github.com/BurntSushi/ripgrep

Gallant, & others. (2025a). *rebar*. https://github.com/BurntSushi/rebar

Gallant, & others. (2025b). *regex* (Version 0.3.2). https://github.com/rust-lang/regex

Hodgman, T. C. (2000). A historical perspective on gene/protein functional assignment. *Bioinformatics*, *16*(1), 10–15. https://doi.org/10.1093/bioinformatics/16.1.10

Katz, K., Shutov, O., Lapoint, R., Kimelman, M., Brister, J. R., & O'Sullivan, C. (2022). The sequence read archive: A decade more of explosive growth. *Nucleic Acids Research*, *50*(D1), D387–D390. https://doi.org/10.1093/nar/gkab1053

Kleene, S. (1951). Representationof events in nerve nets and finite automata. *CE Shannon and J. McCarthy*. https://doi.org/10.1515/9781400882618-002

Merrett, J. E., Nolan, M., Hartman, L., John, N., Flynn, B., Baker, L., Schang, C., McCarthy, D., Lister, D., Cheng, N. N., & others. (2024). Highly sensitive wastewater surveillance of SARS-CoV-2 variants by targeted next-generation amplicon sequencing provides early warning of incursion in victoria, australia. *Applied and Environmental Microbiology*, *90*(8), e01497–23. https://doi.org/10.1128/aem.01497-23

Schlegel, M., & Seyboldt, A. (2025). *seq_io: FASTA and FASTQ parsing and writing in Rust* (Version 0.3.2). https://github.com/markschl/seq_io

Sims, N., & Kasprzyk-Hordern, B. (2020). Future perspectives of wastewater-based epidemiology: Monitoring infectious disease spread and resistance to the community level. *Environment International*, *139*, 105689. https://doi.org/10.1016/j.envint.2020.105689

Valdivia-Granda, W. A. (2012). Biodefense oriented genomic-based pathogen classification systems: Challenges and opportunities. *Journal of Bioterrorism & Biodefense*, *3*(1), 1000113. https://doi.org/10.4172/2157-2526.1000113

Xylogiannopoulos, K. F. (2021). Pattern detection in multiple genome sequences with applications: The case of all SARS-CoV-2 complete variants. *bioRxiv*, 2021–2004. https://doi.org/10.1101/2021.04.14.439840