The Journal of Open Source Software

kmhelpers: A Python toolkit for automated management of genomic indexes

Sébastien Bellenous [1], **Kamil S. Jaron** [2], **and Pierre Peterlongo** [1]

1 Genscale, Univ. Rennes, Inria, CNRS, IRISA - UMR 6074, Rennes, F-35000 France **2** Tree of Life, Wellcome Sanger Institute, Hinxton CB10 1SA, United Kingdom

DOI: 10.21105/joss.11287

Software

- Review
- Repository
- Archive

Editor: Charlotte Soneson

Reviewers:

- @mbhall88
- @anuradhawick

Submitted: 09 July 2026
Published: 18 November 2026

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

Summary

Large-scale genomic sequence search has become a central problem in modern bioinformatics (Karasikov et al., 2025; Marchet, 2024). A widely used family of methods relies on Bloom filters (BFs) (Bloom, 1970) to record, for each indexed sample, which k-mers (words of length k) it contains; a query then reports the set of samples containing each queried k-mer. Building such an index over many samples is not a single operation but a chain of interdependent steps, and each step demands specialist knowledge: sizing each Bloom filter to its sample, configuring the third-party components used internally by `kmindex` (such as the `findere` (Robidou & Peterlongo, 2021) approximate-membership layer), distributing data and computation, bounding peak RAM, and grouping samples into sub-indexes so that the final index stays small without slowing queries. An error at any step can invalidate downstream results, waste significant computation, or produce an oversized index.

We present `kmhelpers`, an open-source Python toolkit that automates this entire workflow for indexes built with `kmindex` (Lemane et al., 2024). It hides the technical decisions listed above behind a command-line interface (CLI) and a matching Python API that cover every stage of the k-mer index lifecycle, from raw samples up to queries, including the federation of independently built indexes into a single queryable resource.

Statement of Need

k-mer-based sequence databases built with tools such as `kmindex` (Lemane et al., 2024) and `kmtricks` (Lemane et al., 2022) answer queries across large sample collections in genomics, metagenomics, and population studies. They were recently used to index 50 petabases of raw sequencing data (Chikhi et al., 2024). Yet a wide gap separates having the indexing software installed from running a reproducible, end-to-end, size-optimized indexing workflow.

Crossing that gap currently requires a user to simultaneously master: Bloom filter sizing as a function of each sample's k-mer cardinality (number of k-mers to be indexed per sample); configuring internal components of `kmindex`; distributing data and computations; controlling the peak memory; and partitioning the samples into sub-indexes that balance index size against query speed. This expertise is not required to obtain *a* functional index but is needed to obtain an *efficient* one that minimizes index size and maximizes query speed under a fixed false-positive-rate constraint.

`kmhelpers` removes that barrier. It makes the complete process of building, updating, and querying a `kmindex` search engine accessible to any group (research teams, sequencing platforms, hospitals, or data-production centers) that holds a potentially large genomic dataset, whether the resulting index is kept private or shared, without repeated manual re-optimization as the dataset grows. This opens to non-specialists both the creation and maintenance of optimized

indexes.

Indexes are rarely static: as new samples accumulate, they must be regularly updated. `kmhelpers` therefore also offers an update feature for easily adding new samples to an existing index.

Formally, the input is a set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of n genomic samples of various sizes; each S_i is either a raw sequencing dataset or a set of assembled sequences. The output is a `kmindex` index subject to two user-defined limits: (1) the maximum allowed false-positive rate, and (2) the maximum number of sub-indexes, each a file storing a set of BFs as a matrix. `kmhelpers` orchestrates every step between input and output, automating the parametrization and the data distribution.

State of the Field

Several tools tackle large-scale sequence search with k-mer-based data structures. `BIGSI` (Bradley et al., 2019) and `COBS` (Bingmann et al., 2019) use compressed Bloom-filter matrices to answer presence/absence queries across collections of sequencing experiments. `HowDeSBT` (Harris & Medvedev, 2020) and `Mantis` (Pandey et al., 2018) rely on sequence Bloom trees and counting quotient filters, respectively. More recent colored-k-mer indexes such as `MetaGraph` (Karasikov et al., 2025) and `Fulgor` (Fan et al., 2024) push scalability further; see Marchet (2024) for a survey. `kmindex`, which `kmhelpers` wraps, builds on the `kmtricks` (Lemane et al., 2022) counting pipeline and is designed for efficient querying of large sequence collections.

`kmhelpers` does not compete with any of these approaches at the algorithmic level. Its contribution lies in correctly parametrizing `kmindex`'s individual components and orchestrating them into a single, size-optimized workflow. `kmhelpers` implements the decisions a `kmindex` expert would otherwise make by hand: `profile` computes Bloom filter sizes and sample-to-sub-index assignments that minimize total index size under a false-positive-rate constraint; `plan` estimates disk and memory requirements before any build starts; and `manage` federates independently built sub-indexes into one logical index. No dedicated automation layer of this kind existed for `kmindex` prior to `kmhelpers`.

Software design

Background: the sub-index structure

A Bloom-filter index stores all BFs of the same size as a single (row-major) matrix in one file. In such a matrix, each column is one BF (one sample) and each row records the presence (1) or absence (0) of a k-mer across all samples sharing that filter size, assuming a common hash function. A complete index is therefore a set of such matrices, each one a *sub-index* holding all BFs of a given size.

The difficulty is that each BF size must match the number of items it stores, here the distinct k-mers of a sample. If all samples had the same number of distinct k-mers, every BF would share one size and a single matrix would suffice. This is the ideal case, where only one file is opened per query and one matrix row gives the answer across all n samples. In practice, sample sizes differ by several orders of magnitude. Sizing every BF for the largest sample wastes enormous storage space. Giving each sample its own single-column matrix minimizes storage but forces a query to open n files (potentially millions), severely slowing down queries. `kmhelpers` takes the middle ground: the user fixes the maximum number of sub-indexes, and the `profile` step chooses the per-sub-index BF size that minimizes total index size under that constraint, before distributing each input sample to its respective sub-index.

The pipeline

`kmhelpers` exposes the index lifecycle as a sequence of commands, as illustrated in Figure 1:

- **`list`** — recursively discovers all samples in a given directory and counts each sample's distinct k-mers using `ntCard` (Mohamadi et al., 2017) (unless the counts are provided by the user).
- **`profile`** — determines the best set of sub-index BF sizes given the user-defined maximum number of sub-indexes and target false-positive rate.
- **`compose`** — assigns each sample to its sub-index and generates the *files-of-files* describing the data origin of each sub-index.
- **`plan`** — validates sample files, available disk space, and memory upfront, and emits ready-to-execute pipeline scripts.
- **`apply`** — builds all sub-indexes by invoking `kmindex`, with span-level and name-level filtering.

For ease of use, steps `list`, `profile`, and `compose` can be grouped under a single command named **`design`**, and steps `plan` and `apply` can be grouped under the **`build`** command.

Once an index is built, `kmhelpers` also answers queries (**`query`**). Multi-step workflows can be described as declarative YAML pipelines (**`pipeline`**) and executed in a single command.

An additional command, **`manage`**, lets users register several distinct indexes (built locally or hosted anywhere accessible) into one logical index, redirecting each query to all registered indexes at query time.

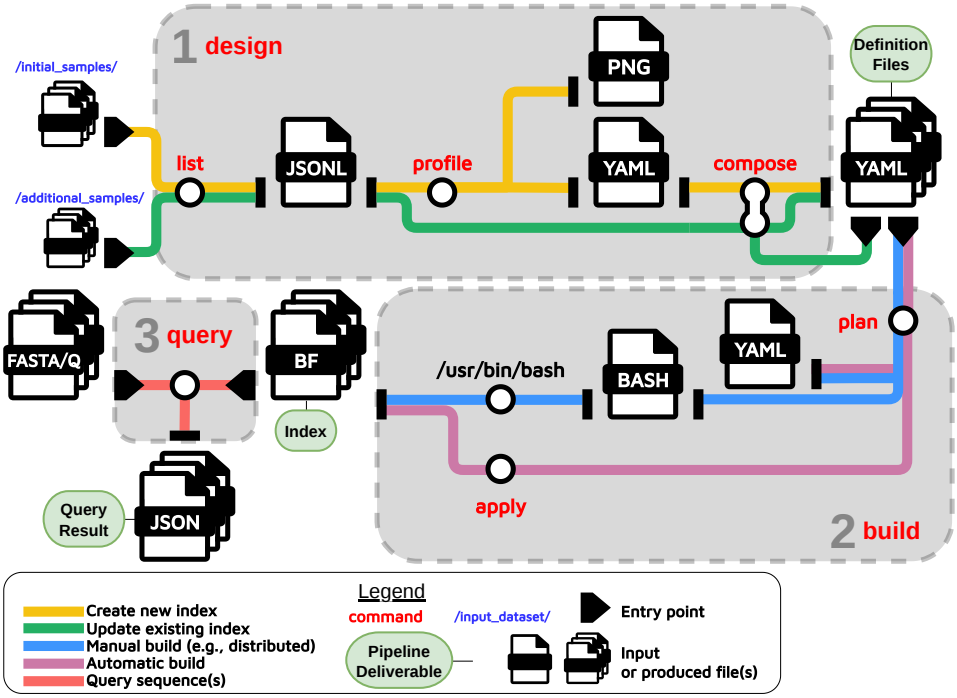


Figure 1: Overview of the `kmhelpers` workflow

Implementation

`kmhelpers` is implemented in Python (≥ 3.10) and distributed via Conda, which installs its bioinformatics dependencies (`kmindex`, `ntCard`) automatically. The CLI is built with `Click` (Ronacher & Click Contributors, 2023), and the package exposes a public Python API

covering all CLI functionality. Together, the declarative YAML index-definition format and the `plan`/`apply` separation make k-mer indexing workflows accessible to researchers who are not experts in the underlying data structures, while remaining flexible enough for large-scale production use.

Research impact statement

Applying `kmhelpers` to the "Tree of Life" dataset

In phase two, the Earth Biogenome Project aims to sequence reference genomes of 150,000 family representatives across the tree of life over the next four years (Blaxter et al., 2025). The Tree of Life Programme at the Wellcome Sanger Institute is one of the major contributors to this effort with large sequencing projects, such as Darwin Tree of Life for all species found in Britain and Ireland (Darwin Tree of Life Project Consortium, 2022). To date, Tree of Life has already released more than 4,000 genomes and is in the process of sequencing an additional 3,000. The project is based on robustly identified specimens, but occasionally long-read datasets and chromatin capture (HiC) sequencing do not match, which might happen due to cryptic species, sequencing hybrid specimens, or mistakes in sample handling either by the collector or during the sequencing process. These cases we refer to as sample swaps.

This project is a perfect use case for applying `kmhelpers`: from input data, here a mixture of raw sequencing reads and assembled genomes, to a final index powering a CLI search engine that is now being used for identification of sample swaps (matching sequencing libraries) at scale. Concretely, `kmhelpers` was applied to 7,394 samples and this project also helped shape the design of `kmhelpers` itself. The index is continuously updated as new sequencing data are progressively added, which is essential for sustaining detection of sample swaps.

Using `kmhelpers` for updating the Logan-Search search engine

The Logan-Search search engine (Chikhi et al., 2024) indexes approximately 50 petabases of sequence data from SRA (Katz et al., 2022). Since its creation, the SRA database size has doubled. The index of Logan-Search will be updated using `kmhelpers`, also optimizing the sub-indexes design.

Availability and documentation

`kmhelpers` is released under the GNU General Public License v3.0 and is available at https://github.com/sebllns/kmhelpers. The full documentation is available at https://sebllns.github.io/kmhelpers/.

Acknowledgements

The authors thank Téo Lemane for developing `kmindex` and for his support in addressing feature requests and issues raised during the development of `kmhelpers`. We acknowledge the GenOuest core facility (https://www.genouest.org) for providing the computing infrastructure. The work was funded by the Inria Challenge "OmicFinder" (https://project.inria.fr/omicfinder/), and by the state funding managed by the French National Research Agency under the France 2030 program [ANR-22-PEAE-0005]. KSJ was funded by Wellcome Trust 220540/Z/20/A.

AI Usage Disclosure

AI assistance was used for constrained tasks (code suggestions and drafting the paper) under strict human review at every stage. AI provider used: Claude (Anthropic, 2025).

References

Bingmann, T., Bradley, P., Gauger, F., & Iqbal, Z. (2019). COBS: A compact bit-sliced signature index. *String Processing and Information Retrieval (SPIRE 2019)*. https://doi.org/10.1007/978-3-030-32686-9_21

Blaxter, M., Lewin, H. A., DiPalma, F., Challis, R., da Silva, M., Durbin, R., Formenti, G., Franz, N., Guigo, R., Harrison, P. W., & others. (2025). The Earth BioGenome Project Phase II: Illuminating the eukaryotic tree of life. *Frontiers in Science*, *3*, 1514835. https://doi.org/10.3389/fsci.2025.1514835

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, *13*(7), 422–426. https://doi.org/10.1145/362686.362692

Bradley, P., den Bakker, H. C., Rocha, E. P. C., McVean, G., & Iqbal, Z. (2019). Ultrafast search of all deposited bacterial and viral genomic data. *Nature Biotechnology*, *37*(2), 152–159. https://doi.org/10.1038/s41587-018-0010-1

Chikhi, R., Lemane, T., Loll-Krippleber, R., Montoliu-Nerin, M., Raffestin, B., Camargo, A. P., Miller, C. J., Fiamenghi, M. B., Agustinho, D. P., Majidian, S., Autric, G., Hugues, M., Lee, J., Faure, R., Curry, K. D., Moura de Sousa, J. A., Rocha, E. P. C., Koslicki, D., Medvedev, P., … Babaian, A. (2024). Logan: Planetary-scale genome assembly surveys life's diversity. *bioRxiv*. https://doi.org/10.1101/2024.07.30.605881

Darwin Tree of Life Project Consortium. (2022). Sequence locally, think globally: The Darwin Tree of Life Project. *Proceedings of the National Academy of Sciences*, *119*(4), e2115642118. https://doi.org/10.1073/pnas.2115642118

Fan, J., Khan, J., Singh, N. P., Pibiri, G. E., & Patro, R. (2024). Fulgor: A fast and compact k-mer index for large-scale matching and color queries. *Algorithms for Molecular Biology*, *19*, 3. https://doi.org/10.1186/s13015-024-00251-9

Harris, R. S., & Medvedev, P. (2020). Improved representation of sequence Bloom trees. *Bioinformatics*, *36*(3), 721–727. https://doi.org/10.1093/bioinformatics/btz662

Karasikov, M., Mustafa, H., Danciu, D., Kulkov, O., Zimmermann, M., Barber, C., Rätsch, G., & Kahles, A. (2025). Efficient and accurate search in petabase-scale sequence repositories. *Nature*, *647*(8091), 1036–1044. https://doi.org/10.1038/s41586-025-09603-w

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

Lemane, T., Lezzoche, N., Lecubin, J., Pelletier, E., Lescot, M., Chikhi, R., & Peterlongo, P. (2024). Indexing and real-time user-friendly queries in terabyte-sized complex genomic datasets with kmindex and ORA. *Nature Computational Science*, *4*(2), 104–109. https://doi.org/10.1038/s43588-024-00596-6

Lemane, T., Medvedev, P., Chikhi, R., & Peterlongo, P. (2022). Kmtricks: Efficient and flexible construction of Bloom filters for large sequencing data collections. *Bioinformatics Advances*, *2*(1), vbac029. https://doi.org/10.1093/bioadv/vbac029

Marchet, C. (2024). Advances in colored k-mer sets: Essentials for the curious. *arXiv Preprint arXiv:2409.05214*. https://doi.org/10.48550/arXiv.2409.05214

Mohamadi, H., Khan, H., & Birol, I. (2017). ntCard: A streaming algorithm for cardinality estimation in genomics data. *Bioinformatics*, *33*(9), 1324–1330. https://doi.org/10.1093/bioinformatics/btw832

Pandey, P., Almodaresi, F., Bender, M. A., Ferdman, M., Johnson, R., & Patro, R. (2018). Mantis: A fast, small, and exact large-scale sequence-search index. *Cell Systems*, *7*(2),

201–207.e4. https://doi.org/10.1016/j.cels.2018.05.021

Robidou, L., & Peterlongo, P. (2021). Findere: Fast and precise approximate membership query. *String Processing and Information Retrieval: 28th International Symposium, SPIRE 2021, Lille, France, October 4–6, 2021, Proceedings 28*, 151–163. https://doi.org/10.1007/978-3-030-86692-1_13

Ronacher, A., & Click Contributors. (2023). *Click: Python composable command line interface toolkit*. https://click.palletsprojects.com