

# bioseqkit: A Lightweight, Dependency-Free Biological Sequence Processing Toolkit

Jilai Cheng

[chengjilai@sjtu.edu.cn](mailto:chengjilai@sjtu.edu.cn)

## Abstract

Biological sequences (DNA, RNA, protein) are the most fundamental objects of study in bioinformatics, and virtually every downstream analysis begins with the reading, statistical characterisation and pre-processing of sequence files. Many practitioners reach directly for high-level libraries such as Biopython and, as a result, never engage with the underlying I/O, streaming and indexing mechanisms. This project presents *bioseqkit*, a lightweight, dependency-free Python package that re-implements these building blocks from scratch: streaming FASTA/FASTQ parsers (with transparent gzip support), sequence statistics, reverse complement and six-frame translation, k-mer and minimizer analysis, and a `samtools faidx`-compatible random-access index. The core library depends only on the Python standard library, follows a modern `src`-layout with a console-script entry point, and is validated by a 39-case `pytest` suite covering all modules. On a 2-Mbp sequence, the multi-process k-mer counter reaches a  $2.9\times$  speed-up on four workers, and the FAI-like index answers a random sub-sequence query in about 0.1 ms after an  $O(n)$  build. The package is a reusable foundation for real sequence-analysis workflows as well as a didactic reference for bioinformatics data-handling design patterns.

**Keywords:** bioinformatics, FASTA/FASTQ, k-mer, minimizer, sequence indexing, Python software engineering

## 1. Background and Motivation

Sequencing technologies produce data at a scale that makes the details of file I/O and memory management first-class concerns. Genome assembly, variant calling, transcriptome quantification and protein-function prediction all rest on the same primitive operations: parsing FASTA/FASTQ files, computing basic statistics, transforming sequences, and extracting sub-sequences efficiently. General-purpose toolkits such as Biopython [1] provide these operations behind convenient abstractions, which is ideal for productivity but opaque for learning. In particular, three ideas central to scalable bioinformatics tooling are easy to overlook: (i) the *iterator/generator* pattern that allows arbitrarily large files to be processed with constant memory; (ii) *k-mer* and *minimizer* [2] sampling, which underpin modern aligners and sketching tools such as `minimap2` [3] and `Mash` [4]; and (iii) *random-access indexing*, exemplified by the `*.fai` index of `SAMtools` [5].

The goal of *bioseqkit* is therefore twofold. First, to build a genuinely useful, installable and testable sequence toolkit. Second, to do so by re-implementing the low-level logic in pure Python – without any bioinformatics dependency – so that the design patterns are made explicit. Typical application scenarios include GC-content and k-mer composition analysis for bacterial species identification, six-frame translation for candidate-ORF screening, and unified feature extraction for downstream machine-learning pipelines.

## 2. Project Design

### 2.1. Architecture

The package adopts a modular `src`-layout. Each module has a single, well-defined responsibility and the public API is re-exported from the package root (Table 1).

Module	Responsibility
io	Streaming FASTA/FASTQ parsers, gzip handling, Phred decoding, FASTA writer
stats	Length distribution, N50, GC content, N-base ratio, base-composition matrix
transform	Reverse complement (IUPAC aware) and six-frame translation
kmer	k-mer counting, top-k, canonical k-mers, parallel counting, minimizers
index	FAI-like index construction and chr:start-end random access
entrez	NCBI E-utilities download helper (standard-library HTTP only)
cli	argparse command-line interface and console-script entry point

Table 1: Module responsibilities of *bioseqkit*.

## 2.2. Core data structures and design principles

Records are represented as frozen dataclass types: `FastaRecord` carries `id`, `description` and `sequence`, while `FastqRecord` additionally holds a `quality` string and can decode Phred scores. Three principles guide the implementation:

1. **Streaming I/O.** Parsers are generators that yield one record at a time, so peak memory is independent of file size. gzip files are detected by extension or magic bytes and opened transparently.
2. **Standard library only.** All algorithms use `collections.Counter`, `str.translate` and `concurrent.futures`; no Biopython or NumPy dependency is introduced in the core.
3. **Correctness by construction.** Reverse complement uses a single translation table over the IUPAC alphabet; six-frame translation reuses the same standard genetic-code table for the three forward and three reverse-complement frames.

## 2.3. Algorithms

*k*-mer counting scans the sequence with a sliding window in  $O(nk)$  time; canonical mode collapses each *k*-mer with its reverse complement. The *parallel* counter splits input into overlapping chunks (overlap  $k - 1$  so that boundary-spanning *k*-mers are preserved), distributes them to a `ProcessPoolExecutor`, and merges the partial counters. *Minimizers* select, for every window of  $w$  consecutive *k*-mers, the lexicographically smallest (canonical) *k*-mer, collapsing consecutive duplicates as in `minimap2` [3]. The *FAI-like index* records, per sequence, the name, length, byte offset of the first base, bases-per-line and bytes-per-line; a query computes the exact byte offset and reads only the required span, giving  $O(1)$  access after an  $O(n)$  one-pass build.

## 2.4. Software engineering

The project ships a PEP 621 `pyproject.toml` (hatchling backend, `bioseqkit` console script), `environment.yml/requirements.txt` for reproducibility, a GitHub Actions workflow running `ruff` and `pytest` on Python 3.10–3.12, and a Sphinx documentation skeleton. Installation is a standard editable install (`pip install -e .`).

## 3. Test Results

### 3.1. Correctness

The `pytest` suite contains 39 tests spread across `io`, `stats`, `transform`, `kmer`, `index` and `cli`, all passing. It covers single- and multi-line FASTA parsing, blank-line and illegal-input boundary cases, FASTQ length-mismatch detection, reverse-complement correctness against known results, six-frame reading-frame coordinates, canonical *k*-mer merging, equivalence of serial and parallel counting, and index round-trip with cross-line random access.

Task	Setting	Time
k-mer count	1 worker	1.98 s
k-mer count	2 workers	1.25 s
k-mer count	4 workers	0.69 s (2.9×)
faidx build	2 Mbp, one pass	0.008 s
faidx fetch	1000 random queries	0.100 s

Table 2: Runtime on a 2-Mbp sequence ( $k = 8$ , canonical).

### 3.2. Performance

Benchmarks were run on a randomly generated 2-Mbp nucleotide sequence with  $k = 8$  canonical k-mers (Table 2). Parallel counting yields a  $2.9\times$  speed-up on four worker processes; the sub-linear scaling reflects process-startup and counter-merge overhead. The FAI-like index builds in a single  $O(n)$  pass and serves random sub-sequence queries in roughly 0.1 ms each.

### 3.3. Demonstration

A Jupyter notebook (`examples/demo.ipynb`) demonstrates the end-to-end workflow: data import (from NCBI [6] with a local fallback), summary statistics, GC-content and length plots, a k-mer spectrum, six-frame translation, the minimizer distribution, indexed random access, and the parallel-versus-serial benchmark.

## 4. Discussion

*bioseqkit* shows that the essential machinery of a sequence toolkit can be rebuilt clearly and correctly on the standard library alone, while remaining fast enough for real work. The exercise makes explicit the streaming, k-mer and indexing patterns that high-level libraries hide.

Several limitations point to future work. The parallel speed-up is bounded by inter-process serialisation and counter merging; a shared-memory or C-backed counting kernel would scale further, and the current process-based design is best suited to CPU-bound work on large inputs. The FAI-like index assumes a plain-text FASTA with uniform line widths and does not yet support the `bgzip/.gzi` random-access scheme used for compressed references. Planned extensions include quality-aware FASTQ statistics, a `minhash/Mash`-style sketching command building on the existing minimizer code [4], and published Sphinx API docs on GitHub Pages. Overall the package meets its minimum deliverables and most advanced goals, and provides a solid, well-tested basis for further bioinformatics tooling.

## Bibliography

- [1] P. J. A. Cock *et al.*, “Biopython: freely available Python tools for computational molecular biology and bioinformatics,” *Bioinformatics*, vol. 25, no. 11, pp. 1422–1423, 2009.
- [2] M. Roberts, W. Hayes, B. R. Hunt, S. M. Mount, and J. A. Yorke, “Reducing storage requirements for biological sequence comparison,” *Bioinformatics*, vol. 20, no. 18, pp. 3363–3369, 2004.
- [3] H. Li, “Minimap2: pairwise alignment for nucleotide sequences,” *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 2018.
- [4] B. D. Ondov *et al.*, “Mash: fast genome and metagenome distance estimation using MinHash,” *Genome Biology*, vol. 17, no. 1, p. 132, 2016.
- [5] H. Li *et al.*, “The Sequence Alignment/Map format and SAMtools,” *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.
- [6] National Center for Biotechnology Information, “NCBI Nucleotide database.” 2024.