FASTA Inspector

Bioinformatics sequence-analysis CLI · automatic DNA/RNA vs. protein classification

Author: Erich Assunção | **Role:** Sole developer — design, implementation, testing & documentation | **Context:** COMP625 Algorithms for Bioinformatics, Athabasca University | **December 2025**

Source code: github.com/eassuncao/fasta_inspector · Python 3.10+ · standard library only · 188 pytest tests

OVERVIEW

Summary

FASTA Inspector is a command-line bioinformatics tool that parses FASTA sequence files, automatically determines whether each sequence is nucleotide (DNA/RNA) or protein, and computes the statistics that matter for that sequence type. It is built to handle **mixed files** — collections that contain both nucleotide and protein records — classifying every sequence independently and applying the right analysis to each. The project spans the full lifecycle: a robust parser, a composition-based type classifier, dedicated DNA and protein metrics engines, formatted reporting, and a 188-test automated suite — delivered with zero external runtime dependencies on the Python standard library alone.

CONTEXT

The problem

FASTA is the lingua franca of sequence data, but a raw FASTA file tells you almost nothing on its own: headers are inconsistent, a single file can interleave genes and their translated proteins, and the statistics you care about are completely different depending on whether a record is nucleotide or protein. Analysts routinely need a fast, dependency-free way to open an arbitrary FASTA file, find out *what is actually in it*, and get sequence-appropriate metrics — without hand-sorting records or reaching for a heavyweight bioinformatics stack.

WHAT I BUILT

The solution

1. A robust FASTA parser

Reads standard FASTA and degrades gracefully on non-standard input: a fallback header-detection path recovers records from files with missing or malformed > headers, so real-world data that would break a naïve parser is still analysed.

2. Automatic sequence-type classification

Each record is classified as DNA/RNA or protein from its residue composition using a 90% nucleotide threshold, letting a single pass sort a heterogeneous file into the correct analysis branch with no manual labelling.

3. Type-specific metrics engines

DNA / RNA metrics

- Sequence length
- GC content (%) and AT/GC ratio
- Ambiguous nucleotide (N) count
- Per-base composition (A, T/U, G, C)

Protein metrics

- Sequence length
- Composition of all 20 standard amino acids
- Estimated molecular weight
- Per-residue percentage distribution

4. Readable reporting

Per-sequence summaries plus a file-level classification summary (total sequences, DNA count, protein count and proportions) formatted for quick reading in the terminal.

INTERFACE

Usage & example output

```
$ python main.py path/to/your/file.fasta      # or just: python main.py  (defaults to test_mixed.fasta)

Processing FASTA file: test_mixed.fasta
...
[ per-sequence type classification and metrics ]
...

=== Classification Summary ===
Total sequences: 13
DNA sequences:     4  (30.8%)
Protein sequences: 9  (69.2%)
```

Running the bundled 13-sequence mixed dataset (4 DNA, 9 protein) used for validation.

ENGINEERING

Architecture & technical highlights

- **Modular, single-responsibility design.** The pipeline is split into focused modules — `fasta_reader` (parsing), `type_classifier` (detection), `dna_metrics` and `protein_metrics` (analysis), `utils` (shared helpers) — orchestrated by `main.py`, so each stage is independently testable and replaceable.
- **Composition-driven classification.** Sequence identity is inferred from residue makeup rather than trusting headers, making the tool resilient to inconsistent or missing FASTA annotations.
- **Graceful fallback parsing.** Explicit handling for headerless and non-standard files rather than failing on the first malformed record.
- **Zero-dependency footprint.** Runs on the Python standard library alone — nothing to install to analyse a file, which keeps it trivially portable and reproducible.
- **Honest scope.** Known limitations (fixed 90% threshold, single-file processing, in-memory loading, console-only output) are documented alongside a concrete roadmap — batch processing, CSV/JSON/TSV export, configurable thresholds, codon-usage and hydrophobicity analysis, a streaming parser for large files, and an optional GUI.

STACK

Technology

Language	Python 3.10+
Runtime dependencies	None — Python standard library only
Domain	Bioinformatics · FASTA parsing · sequence composition analysis · DNA/RNA & protein statistics
Testing	pytest — 188 tests across parser, classifier, DNA metrics, protein metrics, utils & orchestration
Tooling / practices	Git version control · modular architecture · AI-assisted development (GitHub Copilot) with independent implementation, testing & documentation
Interface	Command-line (CLI)

ENGINEERING RIGOR

Quality & delivery

- **188 automated tests.** Six pytest modules cover every component — parsing edge cases, classification thresholds, both metrics engines, shared utilities, and end-to-end orchestration.
- **Reproducible test data.** Curated real-world datasets — *E. coli* coding DNA and its translated proteins, HLA-B27 nucleotide and protein sequences, and a UniProt human complement protein — plus a purpose-built 13-sequence mixed file and dedicated fixtures for headerless / non-standard input.
- **Documented and versioned.** A thorough README covering datasets, features, usage, limitations and a forward roadmap, developed across clear Git milestones from initial modules to finalized documentation.

TAKEAWAYS

Skills demonstrated

Bioinformatics sequence analysis · FASTA parsing & robust input handling · heuristic classification design · algorithm implementation · modular / single-responsibility architecture · dependency-free Python engineering · comprehensive automated testing with pytest · test-data curation · CLI tool design · technical documentation · independent end-to-end ownership.

Academic project — created for COMP625 (Algorithms for Bioinformatics), Athabasca University, December 2025. Full source code, tests and datasets are public at github.com/eassuncao/fasta_inspector.