Analysis and Development of Tools for Genome and Transcriptome Manipulation

Dhananjay Raman[1]

[1]Department of Computer Science and Engineering, IIT Bombay , d.raman@iitb.ac.in

Numerous tools currently exist for manipulating transcriptome data in the form of GTF files, including PyRanges, gffutils, pybedtools, and plyranges. These tools specialize in various types of queries, such as interval arithmetic, aggregate queries, and data filtering, but their performance can vary depending on the specific task. This project aims to systematically benchmark these tools across a set of typical workloads to identify their strengths and weaknesses.
Based on the benchmarking results, tools that underperform in certain workloads are augmented with additional data structures. Such targeted optimizations will be implemented to enhance their efficiency. The goal is to improve the speed, memory usage, and scalability of these tools without compromising accuracy.
The project aims to provide faster, more reliable software for transcriptome analysis, ultimately contributing to more efficient genomic and disease-related research. The improved tools will streamline data processing in high-throughput studies, enabling faster discovery and better handling of large datasets.

Introduction

In the field of computational biology, efficient analysis of genomic and transcriptomic data is critical for advancing our understanding of biology and disease. Genomic annotations, often stored in formats like GTF (General Transfer Format), provide detailed information about genes, transcripts, and other functional elements, enabling downstream applications such as differential gene expression analysis, alternative splicing studies, and genome-wide association studies. Given the increasing size and complexity of transcriptomic datasets generated by high-throughput sequencing technologies, the efficiency and scalability of tools for processing such data have become essential.

A variety of software tools have been developed for manipulating and analyzing transcriptome data, including PyRanges (1), gffutils (2), GRanges (3), and plyranges (4). Each of these tools is tailored to specific types of queries, such as genomic interval operations, aggregate queries, and data filtering. For example, PyRanges excels in genomic interval arithmetic due to its tree-like, while gffutils is highly versatile for managing complex GTF/GFF annotations. Similarly, GRanges and plyranges, built within the R ecosystem, offer advanced genomic data transformation capabilities and seamless integration with other bioinformatics workflows. However, their relative performance—measured in terms of execution time, memory usage, and scalability—can vary widely depending on the task.

Despite their widespread use, the lack of comprehensive benchmarks across typical workloads poses a challenge for researchers. For instance, a tool optimized for interval arithmetic, such as finding overlaps, may underperform in aggregate queries like calculating total exon lengths for a gene. This variation in performance makes it difficult for researchers to select the most appropriate tool for their specific needs without a clear understanding of the trade-offs.

This project addresses these challenges by systematically benchmarking these tools on common operations encountered in transcriptomic data analysis. The benchmarking process includes tasks such as genomic interval arithmetic (e.g., merging, finding overlaps) and aggregate computations (e.g., counting exons, calculating total lengths). The evaluation covers both performance metrics—execution time and memory usage—and scalability on datasets of varying sizes, including the Ensembl Human GTF (GRCh38.p13) and Mouse GTF (GRCm39).

Methods

In this section, we describe the methods used to benchmark the performance of the four genomic tools across three categories: data loading, aggregate queries, and interval queries. The tools tested were plyranges, pyranges, gffutils-pybedtools, and sql-pybedtools.

Performance Metrics

Data Loading. The time taken to load the GTF files (Ensembl Human and Mouse) into the respective data structures for each tool was measured. These data structures vary between tools, ranging from in-memory data frames to SQL databases and in-memory PyBedTools (5) objects. The time for data loading was recorded in seconds to assess the initial overhead of each tool before performing any further analyses.

Aggregate Queries. Aggregate queries refer to operations that summarize or perform calculations on genomic features, such as counting or measuring lengths. Three distinct aggregate queries were designed to test the tools' abilities to efficiently process genomic annotations and summarize data.

1. **Aggregate Query 1**: *Count the number of exons for each gene*
   This query counts the number of exon features that are annotated within each gene. It provides an understanding of the tool's ability to handle feature counts across hierarchical genomic structures. For each gene

in the dataset, the number of exon annotations was calculated, which helps test the tool's efficiency in processing and aggregating feature counts.

2. **Aggregate Query 2**: *Calculate the total length of exons for each gene*
   This query sums the lengths of all exon features within each gene. By measuring the exon lengths for each gene, this test assesses the tool's ability to perform arithmetic operations on genomic intervals, as well as its efficiency in aggregating information from potentially large datasets.

3. **Aggregate Query 3**: *Identify the chromosome with the highest number of transcripts*
   This query identifies the chromosome with the most gene/transcript annotations. It tests the tool's ability to efficiently aggregate data across chromosomes and identify which chromosome contains the highest concentration of annotated features, representing a type of global summary operation.

Interval Queries. Interval queries refer to operations that manipulate genomic intervals directly, such as merging intervals, finding overlaps, and subtracting intervals. These queries are important for testing the tools' ability to perform spatial and temporal operations on genomic annotations.

1. **Interval Query 1**: *Merging Overlapping Exon Intervals*
   This query identifies and merges overlapping exon intervals. Merging intervals is an important operation when dealing with overlapping annotations in genomic data. It tests the tools' ability to handle overlapping intervals and combine them into larger, non-overlapping intervals, which is critical for efficient genomic data manipulation.

2. **Interval Query 2**: *Finding Overlaps with a Specific Interval*
   This query finds all genomic features (e.g., exons, genes) that overlap with a given interval, specified as chr1:100000-200000. This operation tests the tools' ability to perform spatial searches for overlapping genomic features. The ability to find such overlaps is crucial for many genomic analyses, such as identifying potential regulatory regions or regions of interest in a given interval.

3. **Interval Query 3**: *Subtracting Intervals*
   This query subtracts a given set of repetitive regions (chr1:15M-16M,20M-21M) from exon features. Subtracting intervals is useful for removing specific genomic regions from a dataset, such as repetitive regions that may be less informative. This query evaluates the tools' ability to exclude or filter out regions based on specified intervals.

The SQL-PyBedTools Method

A novel method was devised to optimize the handling of aggregate queries and interval operations by combining the strengths of both SQL and PyBedTools. This method involves converting the pyranges data frame into an in-memory SQL database, allowing SQL queries to be sent for performing aggregate operations. After the data is stored in SQL, queries like counting exons or calculating the total length of exons are executed using SQL's built-in aggregation capabilities.
For aggregate queries, SQL's COUNT() and SUM() functions were used to compute the number of exons per gene and the total length of exons per gene. The advantage of using SQL is that it can efficiently handle large-scale aggregation tasks by leveraging optimized database queries, which often perform better than in-memory operations on large datasets.
For interval arithmetic, the sql-pybedtools method converts the pyranges object into a PyBedTools object, which allows for efficient interval operations like merging, finding overlaps, and subtracting intervals. The conversion step involves creating an in-memory PyBedTools object from the SQL database, which then allows the use of PyBedTools' powerful interval arithmetic functions for operations like merging exon intervals and finding overlaps with specific regions.
This hybrid approach provides a balance of performance for both aggregate and interval queries. By offloading the computationally intensive aggregation tasks to SQL and relying on PyBedTools for interval arithmetic, this method is designed to optimize both query speed and memory usage, particularly for large genomic datasets.

Benchmarking Procedure

The performance of each tool was evaluated on both the human (GRCh38.p13) and mouse (GRCm39) Ensembl GTF datasets. For each dataset, the following steps were taken:

1. **Data Loading**: The time taken to load the data into each tool's native data structure was recorded.

2. **Aggregate Queries**: Each of the three aggregate queries (counting exons, calculating exon lengths, and identifying the chromosome with the most transcripts) was executed, and the time taken to complete each query was measured.

3. **Interval Queries**: Each of the three interval queries (merging, finding overlaps, and subtracting intervals) was executed, and the time taken to complete each query was measured.

The execution times for each tool were recorded over multiple runs to account for variability, and results were averaged to provide representative performance metrics for each query and dataset combination.

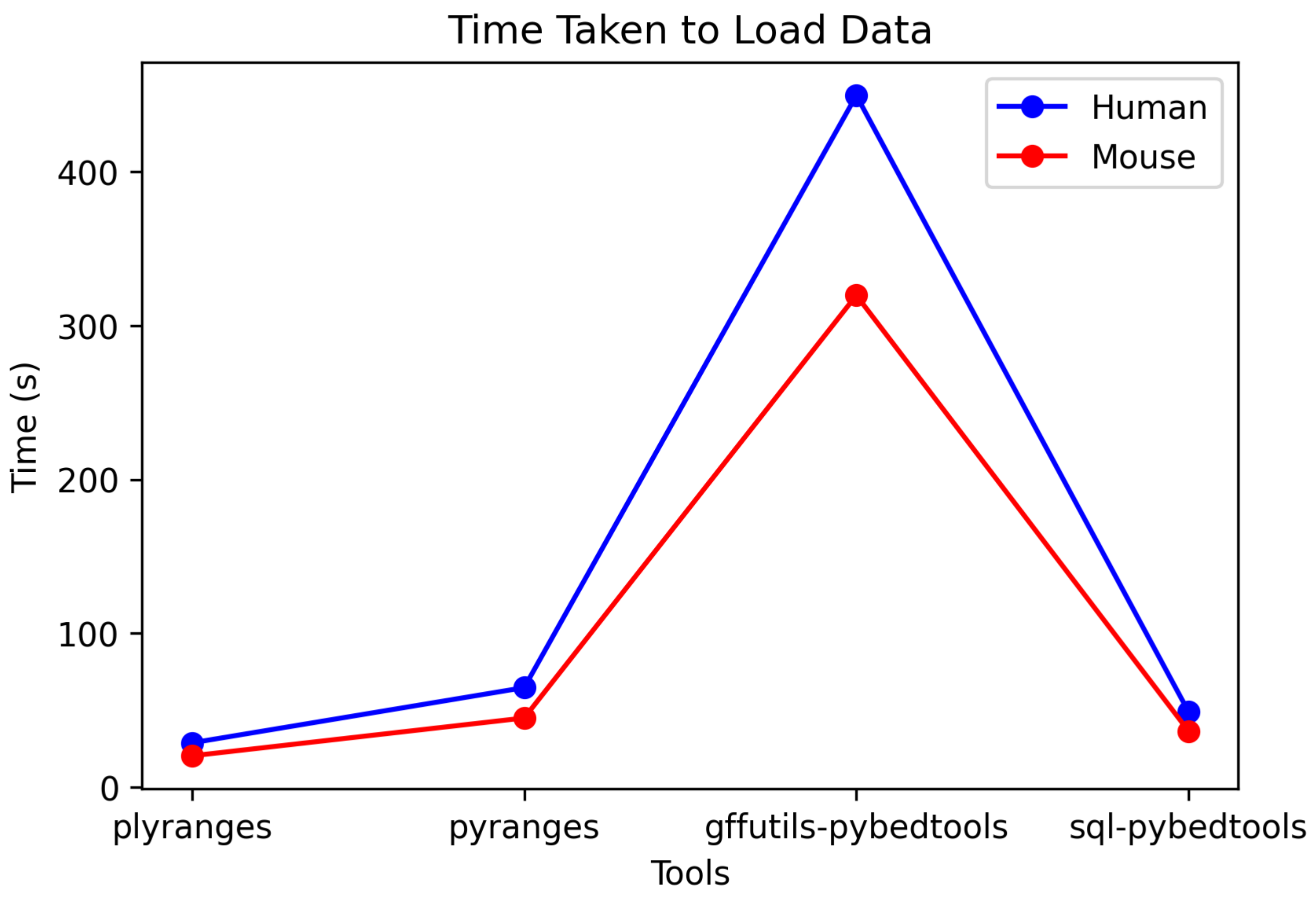

Figure 1. Time taken to load the GTF files for all four tools, for human and mouse datasets

Results

In this section, we present the performance results of the four tools (plyranges, pyranges, gffutils-pybedtools, and sql-pybedtools) across three main categories: data loading, aggregate queries, and interval queries. The results are displayed as plots for both human and mouse datasets.

Time Taken to Load Data

Figure 1 shows the time taken to load the GTF files into the respective data structures for all four tools, comparing the performance between human and mouse datasets.

1. **Human Data**: For the human dataset, plyranges exhibited the fastest data loading time, completing the task in 28.8 seconds. pyranges was slower, taking around 64.7 seconds, while gffutils-pybedtools took a significantly longer 449.5 seconds to load the data. sql-pybedtools demonstrated a comparatively moderate load time of 49.1 seconds.

2. **Mouse Data**: In contrast, the mouse dataset showed a slightly different pattern. plyranges still maintained the fastest load time (20.4 seconds), followed by pyranges at 45 seconds. gffutils-pybedtools took 319.6 seconds, and sql-pybedtools was the fastest among the slower tools, completing the load in 36.5 seconds.

This data demonstrates that plyranges is the most efficient for loading data, while gffutils-pybedtools shows considerable delays in both datasets, likely due to the need for SQL conversion and creating indexes which are not relevant to us.

Time Taken for Aggregate Queries

Figure 2 illustrates the time taken for three different aggregate queries (Query 1, Query 2, and Query 3) for each tool on both the human and mouse datasets.

1. **Human Data**: For aggregate query 1, plyranges took 15.1 seconds, while pyranges performed the task in 5.1 seconds, a notable improvement. gffutils-pybedtools took 44 seconds, significantly lagging behind the others, and sql-pybedtools performed the fastest at 1.4 seconds.

2. **Mouse Data**: Similarly, plyranges and pyranges showed comparable performance on the mouse dataset, with plyranges taking 14.2 seconds and pyranges taking 4.1 seconds for query 1. For query 2, plyranges (14.1 seconds) and pyranges (4.8 seconds) again outperformed gffutils-pybedtools (34.7 seconds), which had a similar performance for both datasets. sql-pybedtools showed the best performance with 1.4 seconds for query 1 and 1.5 seconds for query 2.

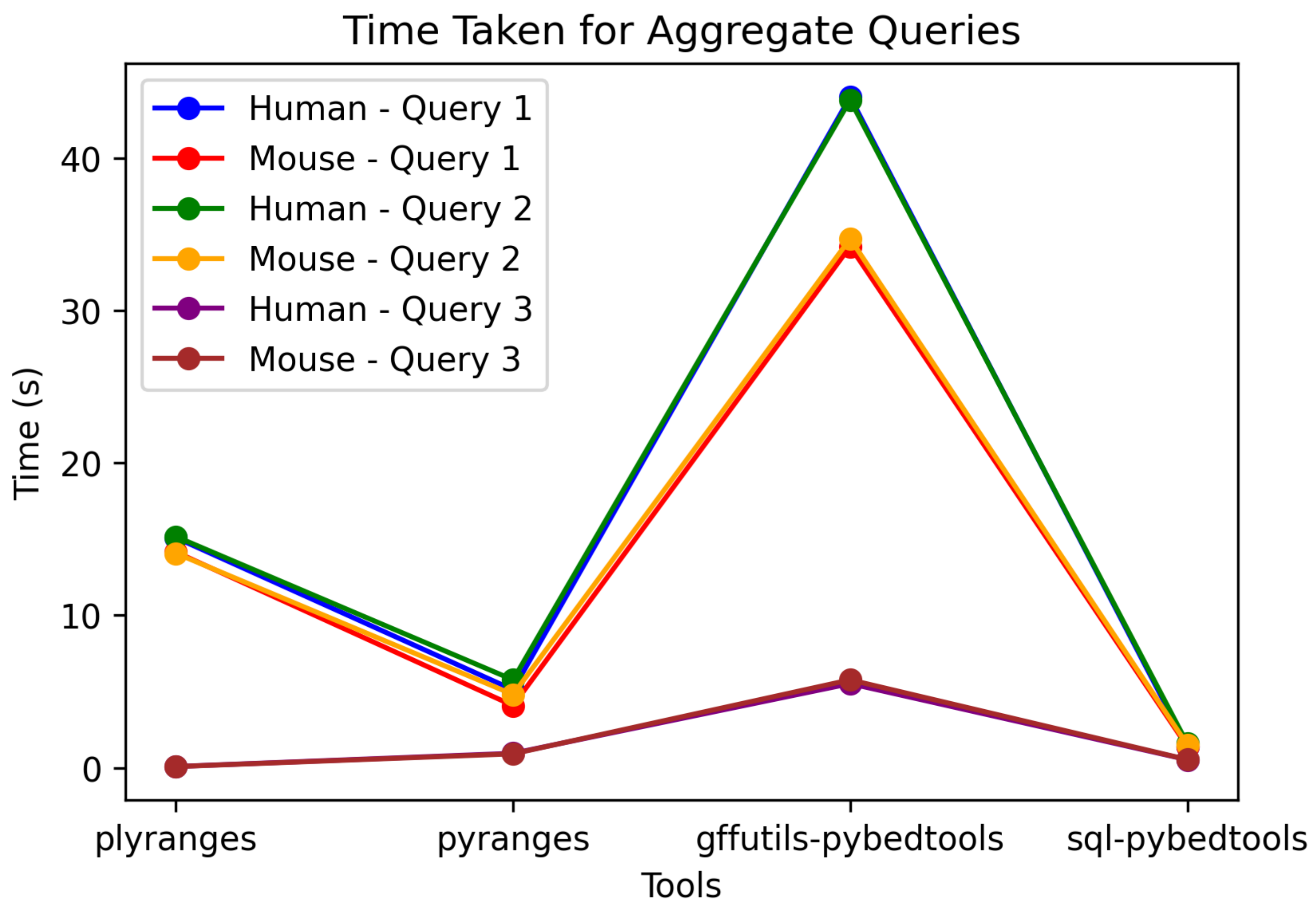

Figure 2. Time taken for different aggregate queries for each tool on both the human and mouse datasets

3. **Query 3**: For the third query, plyranges performed the fastest with 0.085 seconds on the human dataset, followed by pyranges with 0.95 seconds. Both gffutils-pybedtools and sql-pybedtools showed slower times, with gffutils-pybedtools at 5.5 seconds and sql-pybedtools at 0.53 seconds.

Overall, sql-pybedtools was the most efficient tool for aggregate queries, providing the fastest execution times for both datasets and across all queries. plyranges and pyranges were competitive, with plyranges showing consistent performance in handling aggregate queries on the human dataset, while pyranges excelled on the mouse dataset.

Time Taken for Interval Queries

Figure 3 presents the time taken to perform three interval queries (Query 1, Query 2, and Query 3) for each tool on both the human and mouse datasets.

1. **Human Data**: For the first interval query, plyranges was the fastest at 0.17 seconds, followed by pyranges at 1.8 seconds. sql-pybedtools took 6.6 seconds, and gffutils-pybedtools took 60 seconds. For query 2, plyranges remained the fastest at 0.02 seconds, while pyranges took 0.4 seconds, and sql-pybedtools took 7.4 seconds. The second interval query for gffutils-pybedtools again showed significant delays, taking 89 seconds.

2. **Mouse Data**: The interval query results for the mouse dataset mirrored the human results to some extent. plyranges was the fastest for query 1 with 0.12 seconds, and pyranges followed with 3.2 seconds. However, gffutils-pybedtools performed worse on mouse interval queries as well, with 65 seconds for query 1 and 64 seconds for queries 2 and 3. sql-pybedtools showed relatively consistent performance, taking 6.5 seconds for query 1, 6.7 seconds for query 2, and 6.8 seconds for query 3.

In summary, plyranges consistently outperformed other tools for interval queries, particularly for simple queries, while gffutils-pybedtools struggled with longer query times across both datasets. pyranges also provided reasonable performance, especially on the human dataset, but sql-pybedtools was not as efficient for these types of queries, showing slower execution times than the other tools in most cases.

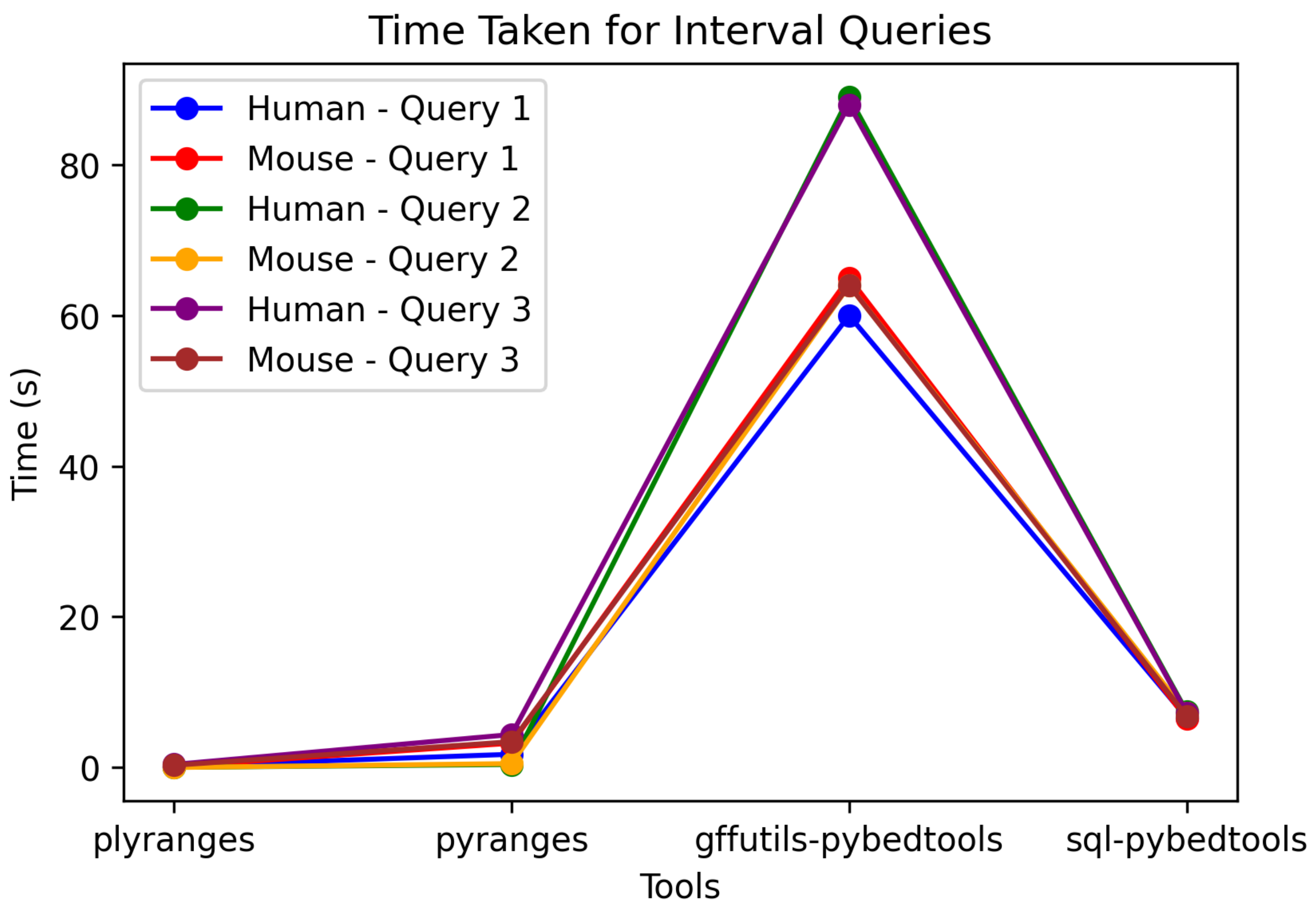

Figure 3. Time taken for different interval queries for each tool on both the human and mouse datasets

Discussion

This benchmarking study evaluated four tools—plyranges, pyranges, gffutils-pybedtools, and sql-pybedtools—across various aggregate and interval queries using human and mouse transcriptome data. Results showed that plyranges excelled in both data loading and interval queries, making it ideal for interval arithmetic tasks. In contrast, sql-pybedtools outperformed the other tools in aggregate queries due to its use of in-memory SQL databases, which allowed for faster execution times. While pyranges performed well in certain tasks, its overall performance was slower compared to plyranges for interval queries and sql-pybedtools for aggregate queries. gffutils-pybedtools showed the slowest performance, especially for interval operations, likely due to the overhead of converting GTF files into SQL databases.

The results suggest that each tool has specific strengths depending on the task. plyranges is well-suited for interval operations, while sql-pybedtools provides faster results for aggregate queries. However, there is potential for improvement in several areas, such as larger dataset scalability, memory usage, and parallelization. Future work could focus on optimizing SQL indexing for interval operations and enhancing parallel processing to improve the overall efficiency of these tools. Despite some performance bottlenecks, the findings highlight the potential for these tools to significantly contribute to genomic data analysis in bioinformatics pipelines.

Availability of materials and data

All code and data is available at github.com/DhanoHacks/DH607-Project-Genomics-Tools-Analysis.

Acknowledgements

The author would like to thank the course instructor for their valuable insights and guidance throughout the project. Additionally, appreciation is extended to the developers of the benchmarking tools for providing accessible resources that made this study possible.

References

1. E. B. Stovner and P. Sætrom, “Pyranges: efficient comparison of genomic intervals in python,” *Bioinformatics*, vol. 36, pp. 918–919, 08 2019.
2. R. Dale, “gffutils: Gff and gtf file manipulation and interconversion,” *GitHub*, 10 2011.
3. M. Lawrence, W. Huber, H. Pagès, P. Aboyoun, M. Carlson, R. Gentleman, M. T. Morgan, and V. J. Carey, “Software for computing and annotating genomic ranges,” *PLOS Computational Biology*, vol. 9, pp. 1–10, 08 2013.
4. S. Lee, D. Cook, and M. Lawrence, “Plyranges: A grammar of genomic data transformation,” *Genome Biology*, vol. 20, 01 2019.

5. R. Dale, “pybedtools: Python wrapper – and more – for bedtools (bioinformatics tools for "genome arithmetic"),” *GitHub*, 05 2010.