Team GeneGorman

Dr. Todd J Treangen, Dr. Fritz Sedlazeck

COMP 416

October 21st, 2024

Genome-Scale Read Mapper

Since the advent of DNA sequencing in 1977, bioinformatics has evolved to meet the growing demand for computational tools to analyze vast genomic datasets. The ability to map millions of short DNA sequences, or "reads," to reference genomes is crucial in answering biological questions and advancing medical research. This report details our team's effort to develop a highly efficient genome-scale read mapper—a software solution designed to handle the massive volume of modern genomic data while maintaining accuracy and performance.

Team Members

Delaney Miller

Gal Kadmon

Henry Tran

Lauren Hu

Yanni (Michelle) Pang

INTRODUCTION

Project Overview:

Read mapping is the process of aligning short DNA sequences, known as "reads," to a reference genome, which can reveal critical genetic differences linked to health and disease. Despite its importance, short-read mapping presents challenges due to repetitive sequences and incomplete references. Our project addresses these challenges by developing a time and memory-efficient algorithm that accurately maps short reads to a genome.

The goal of the Midpoint Report:

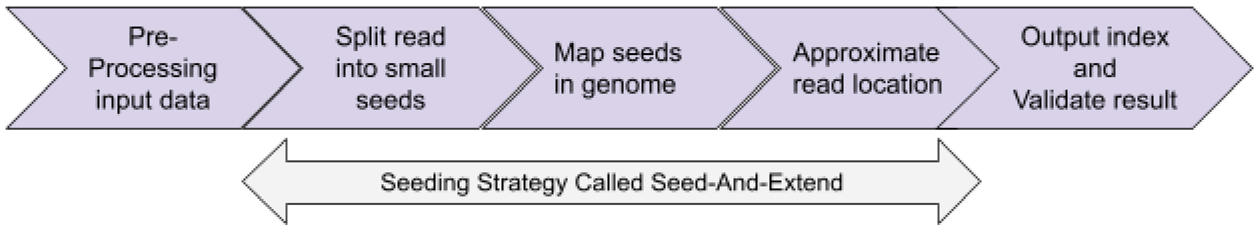
This midpoint report documents our progress in developing key components of the read mapper, including file parsing, read splitting, seeds matching, and localizing best alignments using search algorithms. We will also outline the future steps for implementing more advanced matching techniques and optimizations for the final report.

SYSTEM ARCHITECTURE

High-Level Architecture:

The read mapper system is designed to efficiently align short DNA reads to a reference genome by combining modular components that work together to find the best results. The system begins by preprocessing the input data, constructing an index of the reference genome, and parsing the short reads. It then performs read mapping using a seeding strategy, allowing for precise and rapid alignment. This strategy includes splitting the read into smaller seeds and mapping the seeds to the genome. Once the seeds are mapped, our algorithm uses the indices of the seeds to find the best alignment of the entire read. After the reads are mapped, the system compiles them into a standard output format compatible with bioinformatics tools (SAM file). We also validate the results within our test cases to make sure our algorithm works flawlessly.

High-level Design:


Pre-Processing input data
Split read into small seeds
Map seeds in genome
Approximate read location
Output index and Validate result
Seeding Strategy Called Seed-And-Extend

ROLE ASSIGNMENT

Week / Description	Project Manager	Documentation Lead	Testing Lead	IO Lead	Algorithm Lead
1-3	**Gal:** Management: Led meeting conversations, Created initial grant chart, and assigned ToDos. Code: Wrote a backward search algorithm that uses BWT to find perfect matches.	**Delaney:** Document: Wrote a complete README for the first version of our code including environment download instructions, explanations on organization, background information, and the algorithm for GeneGorman V1 Code: Wrote a BWT helper function to find the transform which was used to find exact read matches.	**Lauren:** Code: Implemented initial BWT approach and wrote tests for early versions of the read mapper	Yanni: Code: Implemented Smith-Waterman approach and integrated it with BWT, then created output in SAM format. Problem: Cannot resolve the bug happens during alignment process, result only half accurate	**Henry:** Code: Swapped out Smith-Waterman approach for a seed-and-extend strategy to improve speed. Problem: Was encountering issues with searching using BWT, so switched to a much simpler suffix array for lookups
3-5	**Delaney:** Management: Led bi-weekly meetings, tracked attendance, tasks, and progress in our original google sheet. Delegated	**Gal:** Document: Wrote SWD report, and updated the ReadMe file initial version.	**Henry:** Code: Implemented the validation module, to validate results against the ground truth	**Lauren:** Code: Wrote unit tests for preprocessing module and for read mapper module	Yanni: Code: Implemented output by grabbing and converting the stored information to SAM format

	tasks to each person each week. Sent updates and action items in our group chat. Created the updated, final version of the Gantt chart used in the presentation. Created the Midterm presentation slides.				

ALGORITHM AND DATA STRUCTURES

Our read mapper system utilizes a combination of efficient algorithms and data structures to align short reads to a reference genome. Each module uses specific techniques that optimize performance and accuracy, as described below:

Suffix Array Construction:

Data Structure: The suffix array is a core data structure used in the Preprocessor module. It stores the starting indices of all suffixes of the reference genome, sorted lexicographically. This allows for efficient searching, as binary search can be performed on the suffix array to quickly locate exact matches for the substrings (seeds).

Seeding and Dividing Reads:

Algorithm: The Read_Mapper module uses a seeding approach to align reads. Instead of trying to match an entire read at once, the read is divided into smaller segments called seeds. Each seed is then searched for in the reference genome using the suffix array. For each found seed, we calculate its potential alignment start. The potential alignment start that exceeds the missing seeds threshold and has the lowest hamming distance is considered the best. This approach increases the chances of finding partial matches even if there are errors or variations in the read.

Data Structure: The seeds are stored as a list, and their corresponding matches are recorded, allowing the mapper to reconstruct the best alignment by analyzing where the seeds align relative to each other.

Binary Search for Exact Matches:

Algorithm: To find exact matches for seeds, the Read_Mapper uses binary search on the suffix array. The find_lower_bound and find_upper_bound methods implement this search by locating the range of indices in the suffix array where the seed appears. This method leverages the sorted nature of the suffix array, ensuring efficient lookup times.

Data Structure: The suffix array, combined with the binary search algorithm, allows the system to handle large reference genomes effectively without scanning the entire sequence.

Reverse Complement Matching:

Algorithm: Since DNA sequences can be read in both forward and reverse directions, the Read_Mapper also aligns the reverse complement of each read. The compute_reverse_complement method generates the complement of a read, allowing the system to detect matches on both strands of the genome.

Alignment Evaluation (Hamming Distance):

Algorithm: The Read_Mapper uses the Hamming distance algorithm to evaluate the similarity between the read and the corresponding segment of the reference genome. This method counts the number of positions where the characters differ, providing a straightforward way to measure how well a read aligns. The compute_hamming_distance function ensures that alignments with fewer mismatches are prioritized.

Data Structure: Alignments are stored in a dictionary, where each read ID is mapped to its alignment details. This makes it easy to retrieve, analyze, and output alignment information.

IMPLEMENTATION DETAILS

Language and Tools:

Our implementation is written in Python. We use the *‘Bio’* library to handle the input data, and the *‘pysam’* and the *‘IO’* libraries to handle the output data.

Modularization:

Our read mapper system is designed to efficiently align reads to a reference genome using a series of modular components. Each module has a distinct role, allowing for easy maintenance, testing, and future enhancements. Below is an overview of the core modules:

1. **Main Module:** The entry point of the system, responsible for directing the entire read mapping workflow. It handles command-line arguments, manages file input/output, initiates preprocessing, and coordinates the read-mapping and validation processes. Additionally, it tracks performance metrics, such as memory usage and processing time, to assess the efficiency of the implementation.
2. **Pre-Processing Module**: The Preprocessor class loads the reference genome from a FASTA file and builds a suffix array for efficient searching. It also reads and stores short reads from FASTQ files, along with their quality scores. This module sets up the necessary data structures to prepare the input for the read mapping process, ensuring that the system can efficiently handle the alignment of multiple reads.
3. **Read Mapper Module:** The Read_Mapper class performs the core alignment of short reads to the reference genome. It uses a seeding approach, where reads are divided into smaller segments (seeds), and exact matches are found using the suffix array. It then finds all potential alignment starts by using these found seeds, and for each potential alignment start, we calculate how many total seeds match to that potential alignment. We then determine the best alignment by finding all potential alignments that pass a certain threshold of required seeds, and select the one with the lowest hamming distance. This is done for both the forward and reverse complement strand of each read to determine the best alignment positions. This modular design allows for flexibility in handling reads that may have errors or variations by allowing for missing seeds and by using hamming distance calculations.
4. **Output Module**: The sam_output module generates a SAM (Sequence Alignment/Map) file containing the results of the read-mapping process. It writes the aligned reads, along with relevant metadata, to a standard format that can be easily processed by other bioinformatics tools. This module ensures that the output is compatible with established pipelines, making it straightforward to integrate the results into larger workflows.
5. **Validation Module:** The Validator module compares the predicted alignments against ground truth data to evaluate the accuracy of the read mapper. It

calculates various performance metrics, including true positives, false positives, and flawed alignments, and provides a summary of the results. Additionally, it tracks resource usage during indexing and mapping, offering insights into the system's overall efficiency. This module is essential for diagnosing potential issues, testing, and improving the alignment process.

User Interface (Command Line Interface):

The user interface for the read mapper system is implemented as a command-line interface (CLI). The CLI allows users to interact with the program by specifying necessary inputs, such as the folder path containing input files.

Input Handling:

Users run the program by specifying a folder path that contains the required input files (reference genome, short reads, and ground truth data). This is done by providing the folder path as an argument when executing the script.

The system validates the folder path and ensures that all required input files are present. If any file is missing or multiple files with the same identifier are detected, the system raises an error with a descriptive message to guide the user.

Command-Line Arguments:

The program is designed to be simple and user-friendly. It accepts one mandatory argument: the path to the input folder. This straightforward design allows users to easily execute the read mapping workflow without needing to manage multiple command-line options.

The system can be extended in the future to support additional arguments, such as options for specifying output file locations, adjusting performance settings, or toggling between different alignment algorithms.

Key Functions By Each Module:

1. Preprocessing Module

The Preprocessor class sets up the necessary data structures and inputs to prepare for efficient read mapping. It handles the initial loading and parsing of the reference genome and short reads.

- **__init__(self, short_reads_ref_file, short_reads_1_file, short_reads_2_file):**
    - **Function**: Initializes the Preprocessor by loading the reference genome from a FASTA file, building a suffix array, and reading short reads from two FASTQ files along with their quality scores.

    - **Contribution**: Sets up essential data structures for the mapping process. The suffix array enables efficient searching, and the parsed reads are stored for easy access during alignment.
- **build_suffix_array(reference_genome)**:
    - **Function**: Constructs a suffix array by sorting the starting indices of all suffixes of the reference genome.
    - **Contribution**: Allows rapid searching for exact matches of substrings (seeds) within the genome, which is crucial for efficient read alignment.
- **get_reads():**
    - **Function**: Retrieves the short reads and their quality scores from the parsed FASTQ data, separating them into two sets. The quality scores represent the confidence level for each read, indicating how accurate the sequencing is at each position.
    - **Contribution**: Provides easy access to the input reads and their associated quality information, allowing the Read_Mapper to process and align them. The quality scores help assess the reliability of the reads during the alignment and help handle the mismatches.

2. Read Mapper Module

The Read_Mapper class performs the core alignment process. It uses the seeding technique described earlier to efficiently map the reads to their position in the reference genome.

- **divide_read_into_seeds(read, num_seeds)**:
    - **Function**: Divides the read into smaller segments (the seeds) to facilitate partial matching.
    - **Contribution**: Increases the chances of aligning reads even if they contain errors or variations by focusing on smaller parts of the read.
- **find_exact_matches(substring)**:
    - **Function**: Uses binary search on the suffix array to find exact matches of a seed in the reference genome.
    - **Contribution**: Efficiently locates regions of the genome that may correspond to parts of the read, enabling faster alignment.
- **align_read_seeding(read, num_seeds, max_missing_seeds)**:
    - **Function**: Attempts to align a read by dividing it into seeds, finding matches for each, and computing possible alignment start locations for each match. Alignment start locations with more seed matches have a higher probability of being a better match. For all start locations that exceed a certain number of seed matches, we take the one with the lowest hamming distance. We do this for both the forward and reverse strand of each read.
    - **Contribution**: Allows for flexible matching by accommodating reads that may have errors or be aligned on the reverse strand, thereby improving overall mapping accuracy.

- **compute_hamming_distance(ref_seq, read_seq)**:
    - **Function**: Calculates the number of mismatches between two sequences.
    - **Contribution**: Helps determine the best alignment by prioritizing matches with fewer differences, improving the reliability of the mapping process.

3. Output Module

The sam_output module handles the formatting and writing of alignment results to a SAM file, ensuring compatibility with standard bioinformatics tools.

- **write_sam_file(sam_filename, read_1_alignments, read_2_alignments)**:
    - **Function**: Writes the results of the read mapping process to a SAM file, including relevant alignment information for each read.
    - **Contribution**: Provides a standardized output format that can be easily used by other bioinformatics tools, enabling further analysis or integration into larger workflows.

4. Validation Module

The Validator class evaluates the accuracy of the read mapping by comparing predicted alignments with known ground truth data.

- **validate(predictions)**:
    - **Function:** Compares the predicted alignments from the read mapper against known ground truth data to evaluate accuracy. For each read, it checks whether the read was correctly or incorrectly aligned based on the starting position provided in the ground truth.
    - **Contribution**: This function plays a critical role in assessing the effectiveness of the read mapping process by providing detailed metrics on alignment accuracy. By distinguishing between different types of alignment errors, it helps identify specific areas where the read mapper is performing well and where improvements are needed. This information is crucial for the remaining time of our project, to refine the alignment algorithms and optimize the overall accuracy of the system.

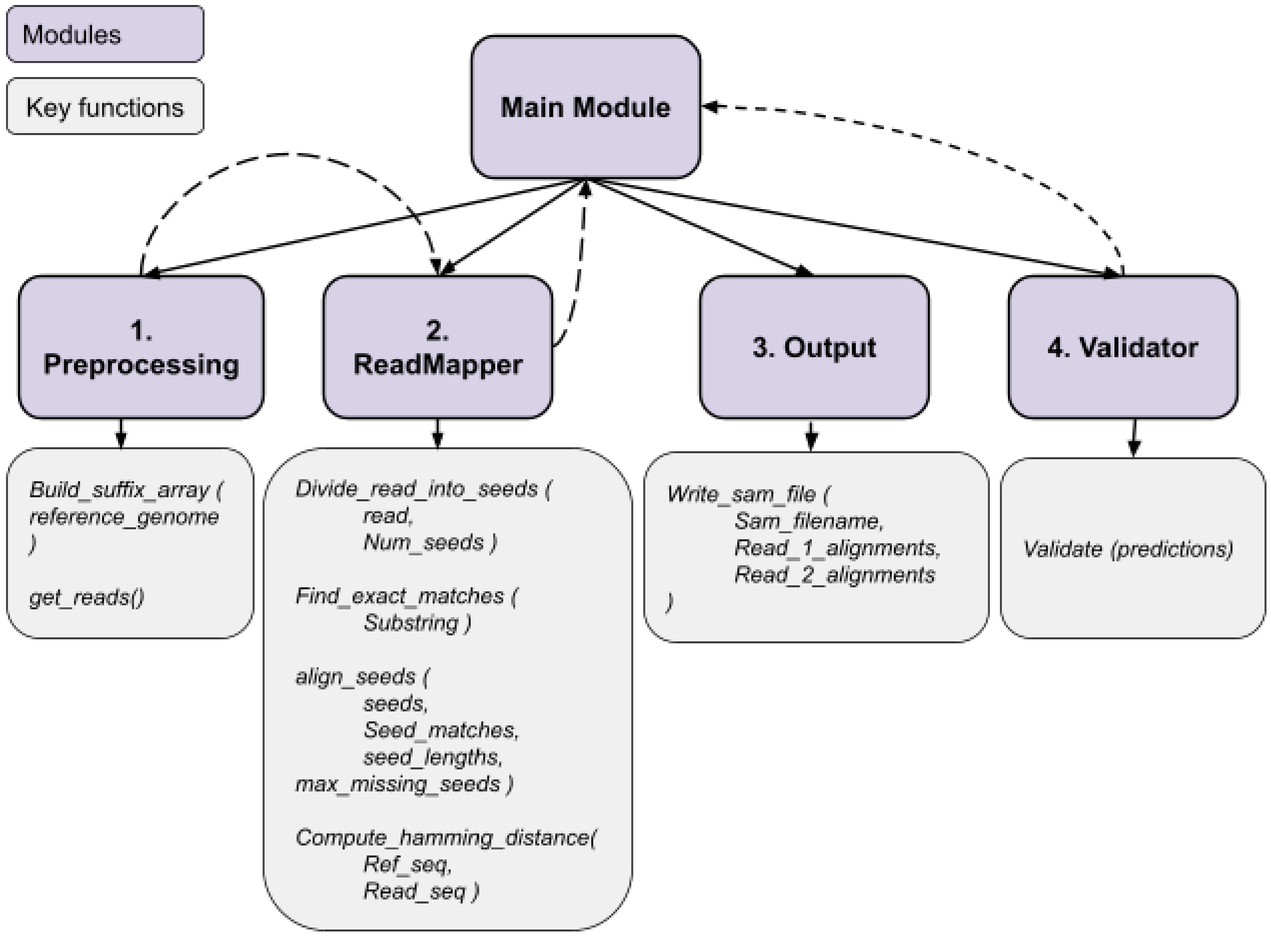
5. Main Module

The main module directs the execution of the entire system, integrating all the components to run the read-mapping process from start to finish.

- **main()**:
    - **Function**: Serves as the central controller handler of the read mapper system, managing the overall workflow from input handling to output generation. The function performs several key tasks:

        - **Argument Parsing:** Ensures that the user provides a valid folder path as a command-line argument, which contains the required input files. It validates the folder path and checks for missing or duplicate files.
        - **Initialization and Execution:** Initializes and coordinates all the necessary components, including preprocessing, read mapping, validation, and output generation. It also tracks memory usage, CPU time, and processing throughput.
    - **Contribution**: It provides a clear entry point to run the program and enables smooth integration of the different modules. The performance metrics collected throughout the process offer valuable insights into the system's behavior, helping us focus on where to optimize our system.

Data Flow Diagram:


Modules
Key functions
Main Module
1. Preprocessing
2. ReadMapper
3. Output
4. Validator
Build_suffix_array (reference_genome)
get_reads()
Divide_read_into_seeds (read, Num_seeds)
Find_exact_matches (Substring)
align_seeds (seeds, Seed_matches, seed_lengths, max_missing_seeds)
Compute_hamming_distance(Ref_seq, Read_seq)
Write_sam_file (Sam_filename, Read_1_alignments, Read_2_alignments)
Validate (predictions)

CHALLENGES AND SOLUTIONS

One of the challenges we faced during the development of our read mapper system was balancing the need for speed and accuracy in the alignment process. Initially, we explored a more advanced approach using the **Burrows-Wheeler Transform (BWT)** and the **Smith-Waterman algorithm**. The idea behind this implementation was to achieve faster lookups for potential matches and allow for the alignment of reads that were not exact matches to the reference genome.

We attempted to implement the **BWT-based** approach to replace our suffix array method. The goal was to improve the efficiency of exact matches, especially when dealing with large genomes. The BWT can significantly reduce the search space, allowing for quick identification of regions where reads might align.

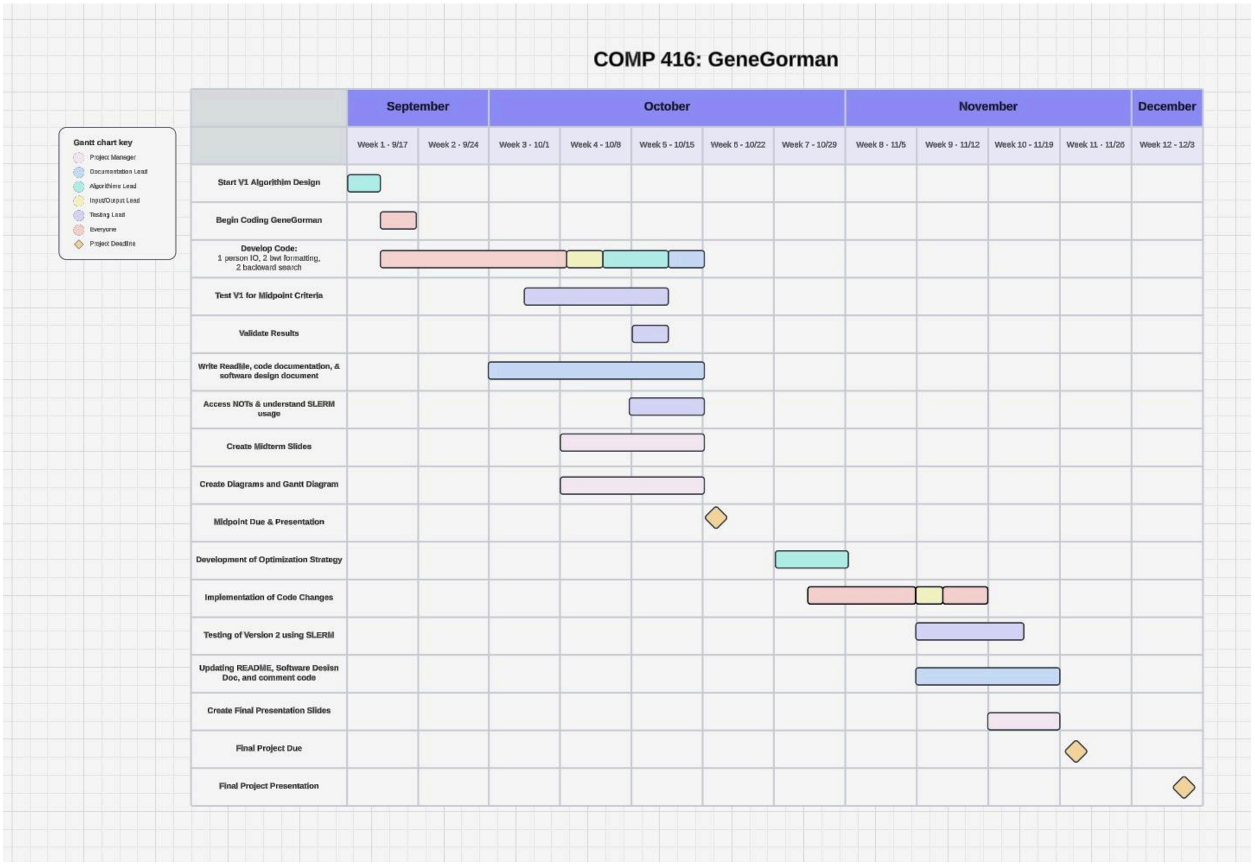
Although the BWT method offered potential advantages in terms of speed, we faced difficulties in integrating it with our existing pipeline. Issues with the implementation prevented us from getting it to function reliably in the timeframe for the midpoint. As a result, we decided to continue with the more straightforward suffix array approach, which was functional and provided an acceptable performance for now.

We Also planned to use the **Smith-Waterman algorithm** to extend the functionality of our read mapper, allowing it to handle reads with small mismatches or gaps. This would have increased the mapper's robustness and improved accuracy in cases where reads were close but not identical to the reference sequence.

Despite its advantages, the Smith-Waterman algorithm is computationally intensive, and integrating it with our seeding approach proved to be more complex than anticipated. The combination of attempting to optimize both the BWT for fast lookups and the Smith-Waterman for alignment introduced implementation errors. Consequently, we opted to focus on a simpler alignment strategy (the hamming distance algorithm) that was quicker to implement and could be further optimized later.

FUTURE WORK

The next steps for our read mapper project involve refining and finalizing the system through a series of tasks mentioned in the following chart. First, we plan to develop an optimization strategy aimed to improve the efficiency and accuracy of our read mapper. Based on this strategy, we will implement code enhancements to address performance bottlenecks and increase scalability. To validate these changes, comprehensive testing will be conducted on the SLURM platform, enabling parallel processing and efficient handling of large datasets. Additionally, we will update the project documentation, including the ReadMe and Software Design Document (SWD), to reflect any changes in the code and system architecture. As the project reaches its final stages, we will prepare presentation slides to effectively communicate our results and findings, along with a final report that documents the implementation, challenges, solutions, and performance of the read mapper. These steps will ensure that the system is robust, well-documented, and ready for deployment or further development.



CONCLUSION

In this report, we have detailed the design, implementation, and evaluation of our read mapper system that efficiently aligns short DNA reads to a reference genome. One of the most important things we cared about when writing the project was to use a system that is built using a modular architecture. This architecture ensures flexibility, scalability, and ease of maintenance. By leveraging data structures like the suffix array and employing the seeding strategy we discussed, the read mapper balances speed and accuracy, allowing it to handle large genomic datasets effectively.

The core components of the system, including preprocessing, read mapping, validation, and output generation, work together under the coordination of the main module. Each module has been designed to perform a specific role, from initializing the necessary data structures and aligning reads to validating results against ground truth data and producing standardized outputs in the SAM format. This modularization not only makes the system robust but also provides a clear path for future enhancements.

Throughout the development, we encountered a few challenges, particularly with trying to implement more ‘advanced’ techniques like the Burrows-Wheeler Transform and the Smith-Waterman algorithm. Although these were not fully integrated, they provided valuable insights into potential areas for future optimization. The current system tracks performance metrics, including memory usage, processing time, and alignment accuracy, offering detailed feedback that helps to identify strengths and areas for improvement.

Looking ahead, we plan to develop and implement an optimization strategy, further refine the code, and conduct comprehensive testing using the SLURM platform. Updates to the project documentation, along with final presentation slides and a detailed report, will ensure that the system is well-documented and ready for deployment or future research.

Overall, for this midpoint report, we have demonstrated a practical and efficient approach to do read mapping, laying the groundwork for continued improvements and further exploration of more advanced techniques. With ongoing development, the read mapper can be enhanced to offer even greater performance and utility in the field of genomic research.

WORKS CITED

Bowe, Alex. “FM-Indexes and Backwards Search.” Alexbowe.Com, alexbowe.com, 22 Feb. 2023, www.alexbowe.com/fm-index/.

Musich R, Cadle-Davidson L, Osier MV. Comparison of Short-Read Sequence Aligners Indicates Strengths and Weaknesses for Biologists to Consider. Front Plant Sci. 2021;12: 657240.

Langmead B, Salzberg SL. Fast gapped-read alignment with Bowtie 2. Nat Methods. 2012;9: 357–359.

Nate. “Short Read Alignment: Seeding - Seven Bridges Genomics.” Seven Bridges, Seven

Bridges, 4 Oct. 2018, www.sevenbridges.com/short-read-alignment-seeding/#:~:text=Alignment%3A%20a%20quick%20review&text=Many%20modern%20alignment%20algorithms%20rely,so%20seeding%20is%20very%20fast.

Langmead, Ben. “Langmead Lab @ JHU - Teaching Materials.” Langmead-Lab.org, 2023, www.langmead-lab.org/teaching.html.

N. Ahmed, K. Bertels and Z. Al-Ars, "A comparison of seed-and-extend techniques in modern DNA read alignment algorithms," 2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Shenzhen, China, 2016, pp. 1421-1428, doi: 10.1109/BIBM.2016.7822731. keywords: {DNA;Computers;Indexes;Genomics;Bioinformatics;Irrigation},