CS249 - Assignment 1: k-mer Index with Minimizer

Mohammad Ashhad

March 26, 2025

1 Introduction

This report details the implementation and analysis of various metagenomic classification approaches for Assignment 1. The task involved classifying DNA sequence reads against a reference database of bacterial genomes using different string matching and indexing techniques. Throughout this assignment, I focused on examining the trade-offs between accuracy, speed, and memory usage across different classification methods.

I worked with five bacterial reference genomes: *Escherichia coli* K-12 MG1655, *Bacillus subtilis* 168, *Pseudomonas aeruginosa* PAO1, *Staphylococcus aureus* NCTC 8325, and *Mycobacterium tuberculosis* H37Rv. For testing, I used two paired-end read datasets (10,000 reads each, 5,000 per R1/R2 file): error-free reads (`simulated_reads_no_errors_10k_R*.fastq`) and error-containing reads (`simulated_reads_miseq_10k_R*.fastq`). All implementations were designed to perform forward-strand-only matching, unless otherwise specified. The full code for implementing the experiments can be found at the following link:
https://github.com/ashhadm/CS_249/tree/main/Week_2

2 Task 1: Metagenome Classification by String Matching

2.1 Task 1.1: Multiple Matches

To handle reads that match multiple organisms, I implemented a Lowest Common Ancestor (LCA) approach. When a read matches multiple reference genomes, I assign it to the lowest common taxonomic node shared by all matching genomes. This approach is biologically meaningful and computationally efficient, as it accounts for conserved genomic regions shared across taxonomic groups while providing appropriate classification specificity.

My taxonomic tree for the five reference genomes is shown below. For example, if a read matches both *E. coli* and *P. aeruginosa*, I assign it to Gammaproteobacteria. If it matches both *B. subtilis* and *S. aureus*, I assign it to Bacilli. In my results, I found that multi-matching reads were most commonly assigned to the Bacteria level, indicating matches across distantly related genomes.

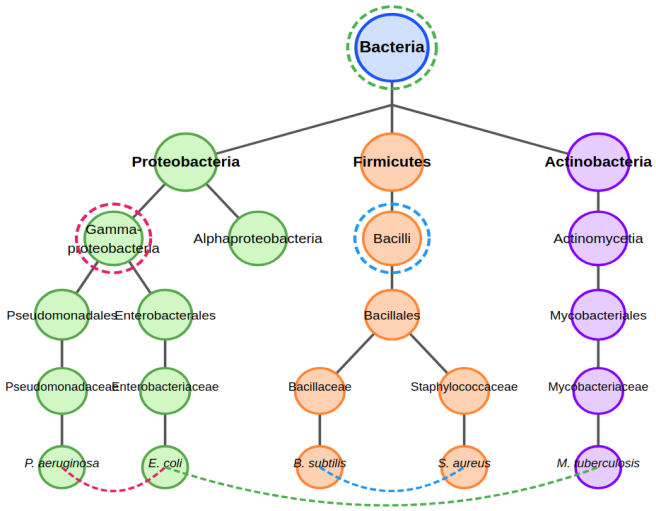

Figure 1: Taxonomic tree for the five reference genomes

2.2 Task 1.2: Exact Matching

I implemented a suffix array-based approach for exact string matching. My implementation used suffix arrays for each reference genome and performed binary search to identify exact matches. To optimize memory usage, I implemented direct character comparison to avoid creating memory-intensive substrings. The matching strategy required the entire read to match exactly on the forward strand only. The implementation processed each read individually against all reference genomes and applied the LCA method for multi-matching reads. Table 1 shows the classification results for both datasets.

Table 1: Classification Results for Suffix Array Exact Matching

Classification	**Error-free(%)**	**Error-containing(%)**
E. coli	30.42	2.14
B. subtilis	5.05	0.48
P. aeruginosa	5.03	0.35
S. aureus	5.03	0.36
M. tuberculosis	5.06	0.42
Multi-match (LCA to Bacteria)	0.09	0.01
Unclassified	49.32	96.24

The performance metrics for my suffix array implementation were: total execution time of 41.00 seconds, index building time of 32.07 seconds, peak memory usage of 1,567.41 MB, and processing speed of 487.75 reads per second. As expected, the exact matching approach performed well on error-free reads but poorly on error-containing reads, since even a single error prevents a match.

2.3 Task 1.4: Comparison with Bioinformatics tools

I compared my suffix array implementation with BLAST (Basic Local Alignment Search Tool), configuring it for exact matching by using `blastn` with parameters set for forward strand searching and 100% identity requirement. Table 2 shows the classification results for BLAST.

Table 2: Classification Results for BLAST

Classification	**Error-free(%)**	**Error-containing(%)**
E. coli	30.61	3.31
B. subtilis	5.07	0.60
P. aeruginosa	5.03	0.50
S. aureus	5.05	0.55
M. tuberculosis	5.04	0.53
Multi-match (LCA to Bacteria/Gammaproteob.)	0.10	0.02
Unclassified	49.10	94.49

BLAST's performance metrics were notably better: total execution time of 5.83 seconds, database creation time of approximately 0.28 seconds, peak memory usage of 255.20 MB, and processing speed of 3,432.53 reads per second. BLAST was approximately 7x faster than my suffix array implementation and used 6x less memory, while showing slightly better sensitivity for error-containing reads. The better performance can be attributed to its highly optimized implementation and years of refinement.

3 Task 2: Metagenomic classification by k-mer index

3.1 Task 2.1: Build the k-mer Index

I implemented a k-mer indexing approach using a k-mer length of 31 (as specified in the assignment). My data structure was a dictionary mapping encoded k-mers to the genomes and positions they occur in. I used 2-bit encoding for DNA bases (A=00, C=01, G=10, T=11) and implemented multi-process k-mer extraction for improved performance.

My k-mer index contained 22,104,664 unique k-mers, which is a tiny fraction of the theoretical k-mer space (4^{31} or 4.6×10^{18}). The k-mers were distributed across genomes as follows: *E. coli* (4,568,108), *B. subtilis* (4,168,371), *P. aeruginosa* (6,221,034), *S. aureus* (2,784,621), *M. tuberculosis* (4,357,767), with just 4,763 k-mers shared between genomes.

The discrepancy between the actual and theoretical k-mer space is due to limited genome sizes compared to the vast theoretical space, biological constraints on DNA sequences, and the exclusion of k-mers containing ambiguous bases ('N').

3.2 Task 2.2: Implement Classification

For k-mer-based classification, I extracted all k-mers from each read, looked them up in the index, counted matches to each reference genome, and assigned the read to the genome with the most matching k-mers. I used the LCA approach for ties or multi-matching k-mers. The classification results are shown in Table 3.

Table 3: Classification Results for K-mer Index

Classification	**Error-free(%)**	**Error-containing(%)**
E. coli	30.45	2.15
B. subtilis	5.05	0.48
P. aeruginosa	5.03	0.35
S. aureus	5.03	0.36
M. tuberculosis	5.06	0.42
Multi-match (LCA to Bacteria)	0.09	0.01
Unclassified	49.29	96.23

The performance metrics for the k-mer approach were: total execution time of 154.50 seconds, index building time of 81.68 seconds, peak memory usage of 8,685.21 MB, and processing speed of 274.68 reads per second. This approach produced nearly identical classification results to the suffix array method, albeit with higher memory usage and slower execution.

3.3 Task 2.3: Minimizers

To reduce memory usage, I implemented a minimizer scheme with k-mer length of 31 and window size (w) of 10 consecutive k-mers. For each window, I selected the lexicographically smallest k-mer as the minimizer. This approach substantially reduced storage requirements.

My minimizer index contained 4,515,766 unique minimizers, a 4.95x reduction from the full k-mer set. These minimizers were distributed across genomes as follows: *E. coli* (942,770), *B. subtilis* (890,253), *P. aeruginosa* (1,238,109), *S. aureus* (573,699), *M. tuberculosis* (870,015), with 920 minimizers shared between genomes. The classification results are shown in Table 4.

Table 4: Classification Results for Minimizer Approach

Classification	**Error-free(%)**	**Error-containing(%)**
E. coli	30.45	2.91
B. subtilis	5.05	0.59
P. aeruginosa	5.03	0.56
S. aureus	5.03	0.49
M. tuberculosis	5.06	0.51
Multi-match (LCA to Bacteria)	0.09	0.01
Unclassified	49.29	94.93

The performance metrics for the minimizer approach were: total execution time of 103.78 seconds, index building time of 79.12 seconds, peak memory usage of 1,917.81 MB, and processing speed of 810.97 reads per second. This approach significantly reduced memory usage compared to the full k-mer index while maintaining similar classification accuracy. It also showed better sensitivity for error-containing reads, possibly because the minimizer selection process naturally provides some error tolerance.

4 Task 3: Real-world data and tools

4.1 Task 3.1: Comparison with Kraken2

I compared my implementations with Kraken2, a widely-used metagenomic classifier. I built a custom Kraken2 database with the same 5 reference genomes and modified parameters to perform more exact-like matching: maximum k-mer length (35 bp), no minimizer spaces, high confidence threshold (1.0), and quick mode for more stringent matching. I processed reads individually (not in paired mode) for direct comparison with my approaches. The classification results are shown in Table 5.

Table 5: Classification Results for Kraken2

Classification	**Error-free Reads (%)**	**Error-containing Reads (%)**
E. coli	59.07	55.34
B. subtilis	9.56	8.93
P. aeruginosa	9.88	9.40
S. aureus	9.45	8.81
M. tuberculosis	9.78	9.18
Unclassified	1.92	7.77

Kraken2's performance metrics were impressive: database building time of 12.43 seconds, peak memory during database building of 7,806.00 MB, classification time of 0.09-0.18 seconds, and peak memory during classification of just 55-70 MB. Kraken2 achieved dramatically higher classification rates than my implementations, was significantly faster (>100x speedup in classification), and showed remarkable robustness to errors (92.23% classified vs 5.07% with my best implementation).

The disparity in performance can be attributed to Kraken2's sophisticated algorithm, which uses a taxonomically-aware approach, employs a weighted classification scheme, can classify based on partial k-mer matches, and has highly optimized data structures and algorithms. Another important thing to note here is that there isn't a way to restrict Kraken2 to forward strand matching only unlike my other implementations where I restricted the implementation to forward strand only, which can further explain the disparity in performance.

4.2 Task 3.2: Real-world Use Case

For this task, I used Kraken2 with the Standard-8 pre-built database to taxonomically classify 10 metagenomic samples: 5 from human gut (SRR11412*) and 5 from wastewater (SRR21907*). The goal was to determine if samples from different environments could be distinguished based on their taxonomic profiles.

I developed a comprehensive analysis pipeline that processed the Kraken2 report files from each sample. The pipeline extracted genus-level taxonomic abundance data (using a minimum abundance threshold of 0.1%), created a taxonomic profile matrix, and applied multiple statistical approaches to analyze environmental differences.

4.2.1 Methodology

The pipeline implemented the following key functions:

- `parse_kraken_report()`: Extracted taxonomic information from Kraken2 reports, filtering by abundance threshold and taxonomic level. For each taxon, it recorded the percentage abundance from column 0, taxonomic rank code from column 3, and cleaned taxon name from column 5.
- `create_taxonomic_profile_matrix()`: Built a samples-by-taxa matrix where each cell contained the abundance percentage of a particular genus in a sample.
- `plot_pca()`: Performed Principal Component Analysis after standardizing the data to identify main sources of variation.
- `perform_hierarchical_clustering()`: Applied hierarchical clustering using Ward's method to group samples based on taxonomic similarity.
- `identify_discriminating_taxa()`: Identified genera with the greatest abundance differences between environments, using Mann-Whitney U tests for statistical validation.

4.2.2 Results

All analysis methods successfully separated the samples into two distinct classes that perfectly corresponded to their environmental origins:

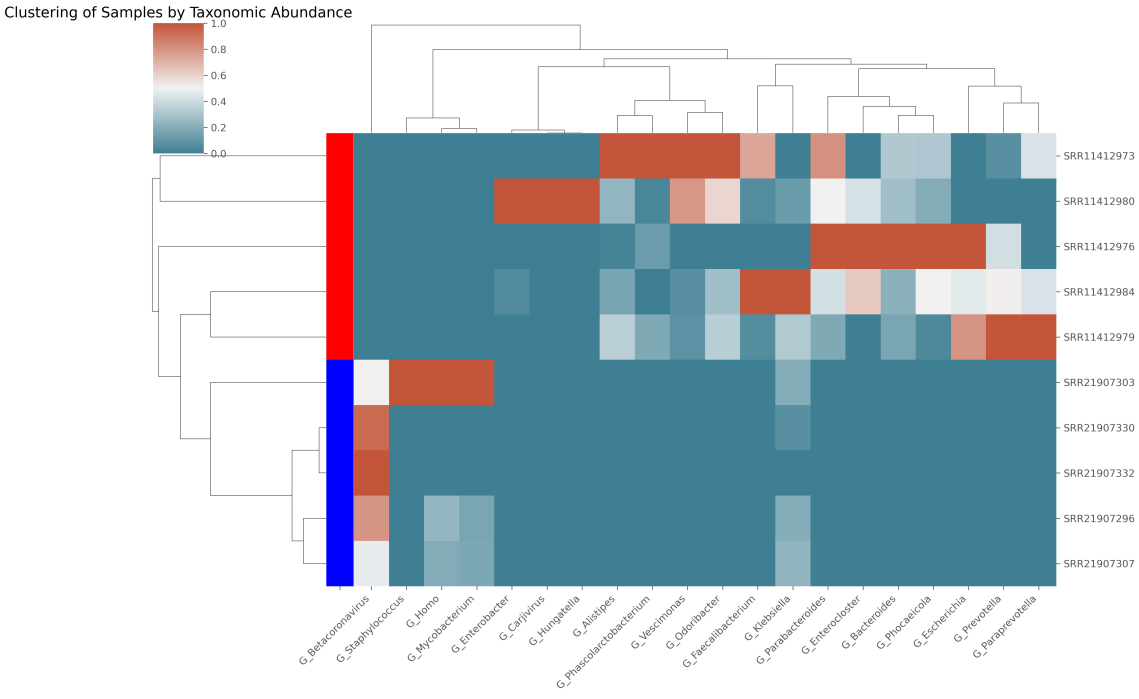

Figure 2: Clustering of samples by taxonomic abundance. The heatmap shows genus-level abundance (standardized) across all samples. Row colors indicate sample source: red for human gut (SRR11412*) and blue for wastewater (SRR21907*) samples. Note the clear clustering by environment and distinctive taxonomic signatures.

The hierarchical clustering analysis (Figure 3) showed a perfect separation between human gut and wastewater samples at the highest level of the dendrogram, indicating substantial differences in their microbial compositions.

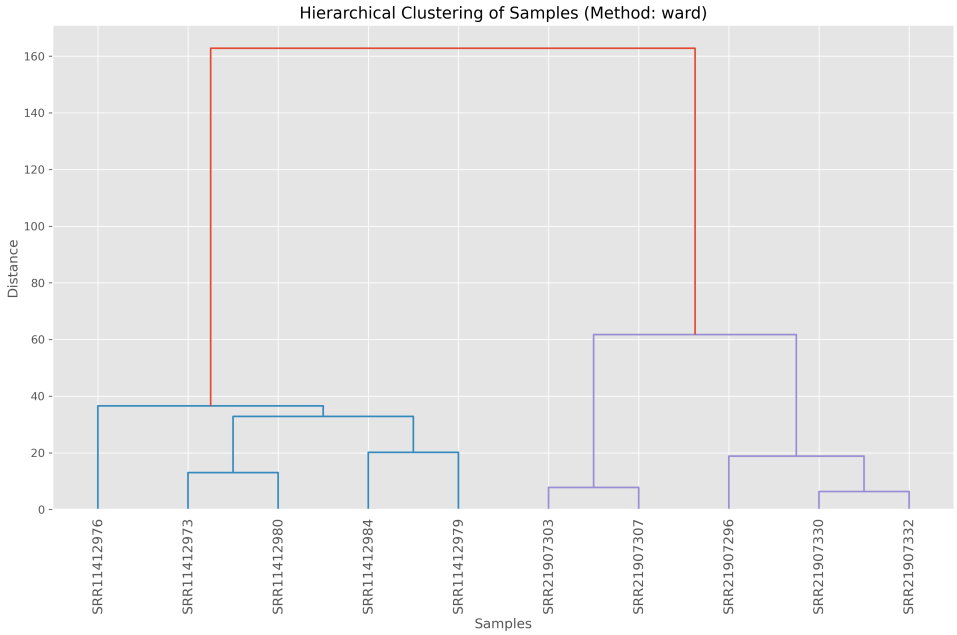

Figure 3: Hierarchical clustering dendrogram using Ward's method. The two primary clusters perfectly separate human gut samples (SRR11412*) from wastewater samples (SRR21907*).

Principal Component Analysis (Figure 4) showed complete separation of samples along both PC1 (30.01% variance) and PC2 (22.93% variance). This strong separation in the first two principal components (explaining over 50% of variance) confirms highly environment-specific taxonomic profiles.

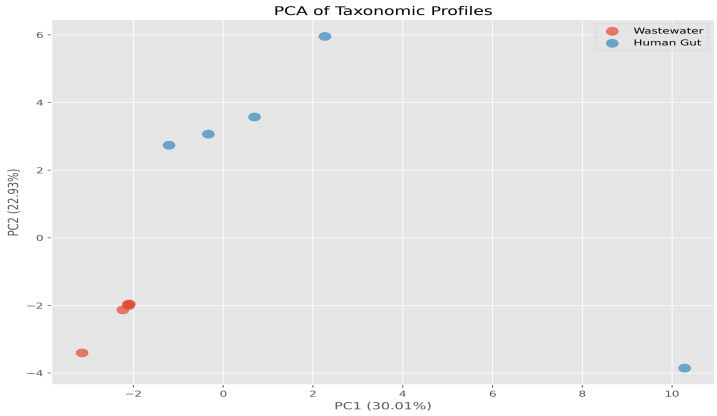

Figure 4: PCA of taxonomic profiles showing clear separation between human gut samples (blue) and wastewater samples (red). The first two principal components explain 52.94% of the total variance.

The analysis identified several genera that strongly differentiate between the two environments (Figure 5):

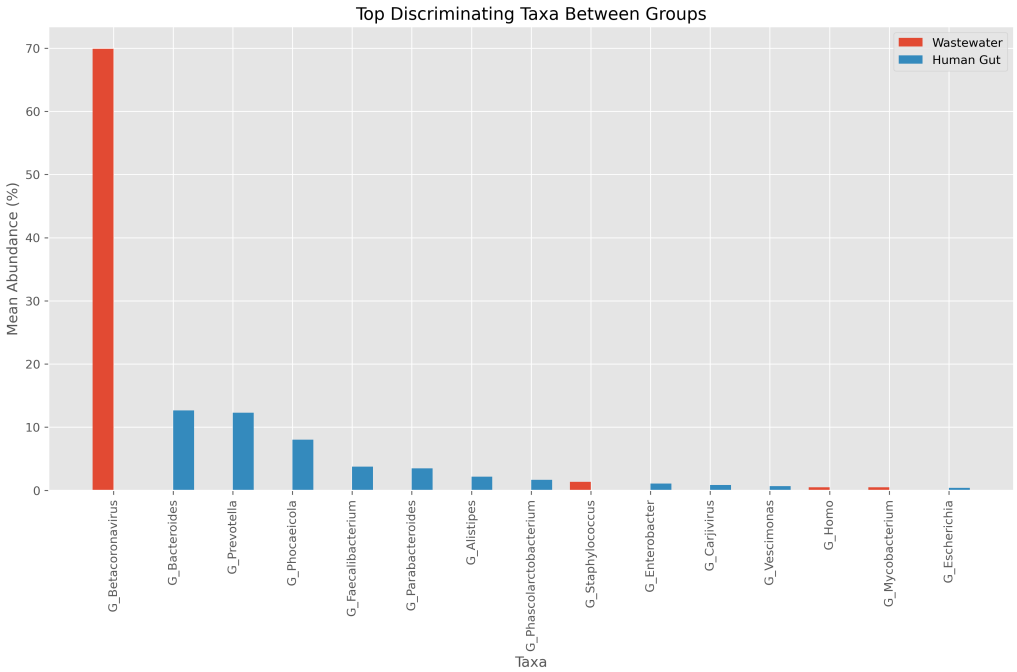

Figure 5: Top discriminating taxa between human gut and wastewater samples. Betacoronavirus shows the most dramatic difference, with high abundance in wastewater samples but virtually absent in gut samples. Bacteroides, Prevotella, and Phocaeicola are characteristic of human gut microbiomes.

The most notable finding was that *Betacoronavirus* showed extreme enrichment in wastewater samples (∼70% abundance) but was virtually absent in gut samples. Human gut samples were dominated by well-known gut microbiota members, including *Bacteroides*, *Prevotella*, *Phocaeicola*, *Faecalibacterium*, and *Parabacteroides*—specialized anaerobes involved in carbohydrate fermentation and butyrate production.

This clear separation reflects fundamental biological differences between these environments: human gut microbiomes are dominated by obligate anaerobes specialized for fermenting complex carbohydrates, while wastewater samples show higher abundance of environmentally resilient bacteria and potential pathogens, with *Betacoronavirus* being particularly notable.

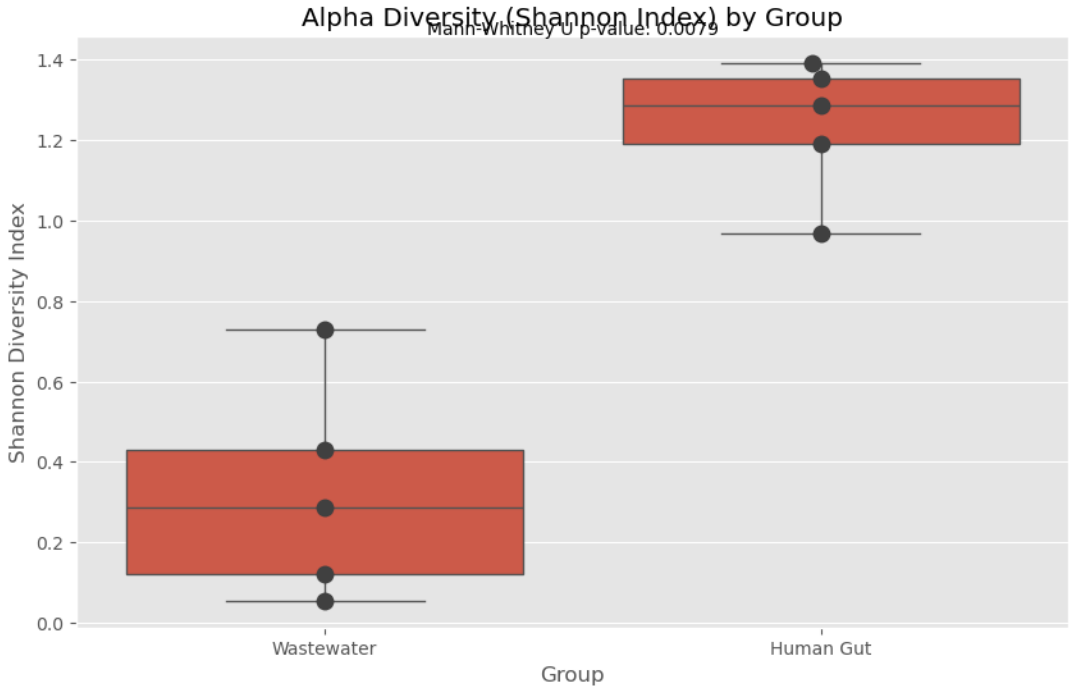

Figure 6: Alpha diversity (Shannon Index) comparison between human gut and wastewater samples. The boxplot shows significantly higher microbial diversity in human gut samples compared to wastewater samples (Mann-Whitney U p-value: 0.0079).

Alpha diversity analysis (Figure 6) revealed significant differences in microbial community complexity between the two environments. Human gut samples exhibited substantially higher Shannon diversity indices (mean ≈ 1.2) compared to wastewater samples (mean ≈ 0.3), with a statistically significant difference confirmed by Mann-Whitney U test ($p = 0.0079$). This higher diversity in gut microbiomes reflects the complex, specialized microbial communities that have co-evolved with human hosts to perform various metabolic functions. In contrast, the lower diversity in wastewater samples suggests dominance by fewer, more abundant taxa (particularly *Betacoronavirus* as identified in our discriminating taxa analysis), potentially indicating more transient or environmentally-driven community structures. The gut environment appears to maintain a more balanced and species-rich ecosystem, while wastewater communities show greater unevenness in their taxonomic distributions.

5 Comparative Analysis of All Methods

Table 6 provides a side-by-side comparison of all implemented methods across key performance metrics.

Table 6: Performance Comparison of All Implementations

Metric	**Suffix Array**	**BLAST**	**K-mer Index**	**Minimizer**	**Kraken2**
Peak Memory (MB)	1,567.41	255.20	8,685.21	1,917.81	7,806.00*
Total Execution (s)	41.00	5.83	154.50	103.78	12.70
Index Building (s)	32.07	0.28	81.68	79.12	12.43
Processing Speed	487.75	3,432.53	274.68	810.97	>100,000
Error-free Classified	50.68%	50.90%	50.71%	50.71%	98.08%
Error Reads Classified	3.76%	5.51%	3.77%	5.07%	92.23%

*For Kraken2, peak memory is during database building; classification used only 55-70 MB.

6 Conclusion

This assignment allowed us to explore different approaches to metagenomic classification, from basic string matching to advanced k-mer indexing techniques. Each method presented distinct trade-offs. The suffix array approach provided balanced memory usage with moderate performance and straightforward implementation. BLAST demonstrated excellent speed and memory efficiency, leveraging decades of optimization. While conceptually simple, the k-mer index proved memory-intensive and showed limited error tolerance. The minimizer approach significantly reduced memory requirements while maintaining accuracy. Kraken2 proved superior in nearly all aspects, demonstrating the value of specialized algorithms.

Key insights from my implementations include the limitation of forward-strand-only matching in real-world applications, as DNA fragments can come from either strand. Error tolerance is crucial since real sequencing data always contains errors. My LCA approach successfully handled ambiguous classifications by assigning multi-matching reads to higher taxonomic nodes (most commonly the *Bacteria* level). Memory efficiency becomes critical when scaling to larger reference databases, and specialized tools like Kraken2 incorporate domain knowledge that significantly enhances performance. The real-world use case analysis demonstrated the power of k-mer-based metagenomic classification for environmental monitoring. Using Kraken2's classification output, we were able to clearly separate human gut and wastewater samples based on their taxonomic profiles. Multiple statistical approaches (hierarchical clustering, PCA, and taxonomic heatmaps) consistently confirmed the strong environmental signature in metagenomic data. The identification of discriminating taxa like *Betacoronavirus* (highly abundant in wastewater) and *Bacteroides*/*Prevotella* (characteristic of gut microbiomes) further validated the effectiveness of this approach for environmental classification.

In future work, I could explore approximate matching techniques to improve error tolerance, implement more sophisticated minimizer schemes, or develop methods that better handle the natural variations in DNA sequences. Additionally, expanding the environmental analysis to include more diverse sample types could yield insights into microbial adaptation across different ecological niches.

A Implementation Hyperparameters

This appendix provides detailed information about the hyperparameters and configuration settings used in each implementation.

A.1 Task 1.2: Suffix Array Implementation

Table 7: Suffix Array Implementation Parameters

Parameter	**Value**
Binary search implementation	Direct character comparison
Character comparison limit	First 100 characters
Number of parallel processes	8
Read chunk size	Dynamic (total reads / num processes)
Memory optimization	No substring creation
Taxonomic assignment strategy	Lowest Common Ancestor (LCA)

A.2 Task 1.4: BLAST Comparison

Table 8: BLAST Configuration Parameters

Parameter	**Value**
BLAST program	blastn
E-value threshold	1e-5
Maximum number of mismatches	0 (Task 1.4), 1 (Task 1.3)
Percent identity threshold	100% for exact match
Number of threads	8
Maximum target sequences	5
Strand search	Plus (forward) only
Output format	Tabular (outfmt 6)

A.3 Task 2.1-2.2: K-mer Index Implementation

Table 9: K-mer Index Parameters

Parameter	**Value**
K-mer length (k)	31 bp
DNA encoding scheme	2-bit encoding (A=00, C=01, G=10, T=11)
Chunk size for parallel processing	1,000,000 bp
Number of parallel processes	8
Minimum k-mer match fraction	1.0 (100%)
Read chunk size	Dynamic (total reads / num processes)
K-mers with ambiguous bases ('N')	Excluded

A.4 Task 2.3: Minimizer Implementation

Table 10: Minimizer Parameters

Parameter	Value
K-mer length (k)	31 bp
Window size (w)	10 consecutive k-mers
Minimizer selection criteria	Lexicographically smallest k-mer
DNA encoding scheme	2-bit encoding (A=00, C=01, G=10, T=11)
Chunk size for parallel processing	1,000,000 bp
Number of parallel processes	8
Minimum minimizer match fraction	1.0 (100%)
Window span	w + k - 1 = 40 bp
Minimizers with ambiguous bases ('N')	Excluded

A.5 Task 3.1: Kraken2 Comparison

Table 11: Kraken2 Parameters for Comparison Task

Parameter	Value
K-mer length	35 bp (maximum)
Minimizer length	35 bp
Minimizer spaces	0 (no spaces, for more exact-like matching)
Confidence threshold	1.0 (maximum)
Classification mode	Quick mode (more stringent matching)
Number of threads	8
Read processing	Individual (not paired)

A.6 Task 3.2: Real-world Use Case

Table 12: Real-world Analysis Parameters

Parameter	Value
Kraken2 database	Standard-8 pre-built database
Number of threads	8
Taxonomic level for analysis	Genus level ('G')
Minimum abundance threshold	0.1%
PCA components	2
Hierarchical clustering method	Ward's method
Distance metric	Euclidean
Top discriminating taxa	15
Heatmap scaling	Standard scale (z-score)
Statistical test	Mann-Whitney U test (two-sided)

B Computational Environment

All experiments were run on a workstation with the following specifications:

Table 13: Computational Environment

Component	**Specification**
Operating System	Linux Ubuntu 22.04 LTS
CPU	AMD Ryzen Threadripper PRO 5975WX s (64)
RAM	128 GB DDR4
Storage	1 TB SSD
Python version	3.8.10
BLAST+ version	2.12.0
Kraken2 version	2.1.2

The implementation code is written in Python, utilizing various libraries including NumPy, Pandas, SciPy, and Matplotlib. Shell scripts were used for Kraken2 execution and SRA data downloading. All multi-threaded operations were limited to 8 threads to ensure consistent performance measurement across different implementations.