Development of a Pipeline for the Benchmarking of Next-Generation Sequencing Quality Control Tools

Final thesis

for obtaining the academic degree
Bachelor of Science (B.Sc.)

at the

Hochschule für Technik und Wirtschaft Berlin
(Berlin University of Applied Sciences)

First examiner:	Prof. Piotr Wojciech Dabrowski
Second examiner:	Prof. Hermann Heßling
Submitted by:	Efim Shliamin
Student number:	s0573270
Course of study:	Applied Computer Science
Date of submission:	04.02.2024

Executive Summary

The thesis delves into the impact of various genome trimming strategies on the genome assembly quality of Mycobacterium tuberculosis, a pathogen of high public health importance [JBA18]. In the era of Next-Generation Sequencing (NGS), achieving accurate genome assembly is vital for understanding genetic diversity, pathogenicity, and drug resistance. The trimming process, a crucial preprocessing step in NGS, involves removing low-quality bases and adapter sequences to improve data quality. This study evaluates the impact of various trimming techniques on genome assembly metrics. It focuses on creating a series of genomic assemblies for Mycobacterium tuberculosis, relevant to the genomic studies of other organisms as well, by utilizing trimming, genome assembling, and genome assembly evaluation processes within a comprehensive software pipeline. The objective is to fine-tune the preprocessing steps to boost the reliability of genomic assemblies. The research employs parallel processing, which is both effective and scalable [VS17], to achieve this goal. Through multiple iterations and creating dozens of genomes, the research uncovers a significant finding: the effectiveness of trimming strategies varies. The primary challenge is to minimize undefined nucleotides for continuous genome assembly while maintaining the quality metrics of it. This insight is crucial for selecting trimming methods that not only improve the genome assembly process but also preserve the integrity of the genomic data. It was found that the quality of some trimming methods significantly surpassed others.

Acknowledgements

I am most grateful to Prof. Piotr Wojciech Dabrowski, whose guidance, support, and instruction have been key throughout the journey of this research. His insight has been invaluable, and his guidance has been the foundation on which this work has been built.

My sincere gratitude goes to Prof. Alexei Gurevich, whose dissertation "Computational Methods for Analysis of Error-Prone Metabologenomic Data" not only inspired my research but was also an integral part of the development of the software used in this thesis.

A special acknowledgment goes to my parents. Their sacrifices and love have not gone unnoticed and have been the quiet, constant force that has fueled my resolve, especially through the unique challenges of my studies abroad.

Contents

1 Introduction

1.1 Background

In the Next-Generation Sequencing (NGS) era, accurate genome assembly is crucial [Pat15] [RB21] [Wan17] [JP04] for understanding organisms like Mycobacterium tuberculosis, including their genetic diversity and drug resistance [Heu21] [Net22] [STG21]. trimming, the process of removing low-quality bases and adapters, is vital for data quality and effective genome assembly. The next paragraphs summarize studies on the impact of trimming algorithms on genome assembly quality and the importance of tailored trimming strategies for optimal data analysis.

1.2 Evaluation of Trimming Algorithms

Nine trimming algorithms were assessed across four datasets for their influence on genome assembly, RNA mapping, and genotyping in NGS data [Del13]. The "SeqPurge: highly-sensitive adapter trimming for paired-end NGS data" article [SSB16] concluded that SeqPurge excels in adapter trimming for paired-end sequencing data due to its high sensitivity, error tolerance, and efficiency, surpassing other tools in RNA mapping and genome assembly scenarios. Sewe et al. [SO22] demonstrated how trimming enhances the quality of genomes assembled from NGS reads, using the optimized tool Trimmomatic to compare the quality of transcriptomes from both trimmed and untrimmed reads.

Liao et al. [Lia20] described the PE-Trimmer algorithm, a trimming tool for NGS sequencing data, highlighting its superior results compared to other trimmers, though not explicitly discussing the pros and cons of the trimming process. Wagner et al. [Wag21]

found that quality-based and fixed-width trimming could improve SNP analysis in enteric pathogen outbreaks in NGS data, suggesting different strategies may be required for genome assembly. Yang et al. [Yan19] observed that trimming low-quality bases from sequencing reads led to shorter scaffolds but saved computational time, without significantly impacting genome assembly completeness. This suggests a need for specific trimming strategies based on the intended use of Assembled genomes.

1.3 Challenges in NGS Data

NGS data can present issues like low-quality fragments, adapters, duplicated reads, contamination, overlaps, varying reads lengths, homopolymeric regions, and sequencing errors. trimming helps mitigate these issues, improving genome assembly quality and analysis accuracy. However, trimming can also result in data loss, necessitating a balance between data preservation and quality.

Our study evaluates the impact of various trimming methods on Mycobacterium tuberculosis genome assembly metrics. We use a robust software pipeline, including fastp [Ope23], SPAdes [Lab22b], and QUAST [Lab22a], running through the Nextflow framework [Tea24a] for parallel processing. Our methodology, comparing trimmed and untrimmed datasets, seeks to understand trimming's effect on genome assembly quality, aiding in selecting methods that balance efficiency with genomic integrity for Mycobacterium tuberculosis and other organisms.

For data analysis and visualization, we utilized Jupyter Notebook [Con24d] integrated with the Pandas [Tea24b] and NumPy [Con24b] libraries. Pandas was instrumental in data manipulation, while NumPy was pivotal in computational tasks, particularly with large numbers. Matplotlib [Con24a] and Seaborn [Con24e] greatly contributed to creating a variety of visualizations, ranging from static graphs to interactive ones.

Plotly [Con24c] further enhanced our toolbox with interactive, web-based illustrations essential for examining genome assembly metrics. This combination of tools in Jupyter Notebook played a significant role in effectively assessing and refining our software pipeline, having substantial implications for the future of genomics research.

2 Theoretical Fundamentals

2.1 Introduction to Molecular Biology of DNA

The DNA (Figure 2.1), the fundamental molecule of life, encodes the genetic instructions (Figure 2.2) vital for the development and functioning of living organisms. As detailed in "Molecular Biology of the Gene" by James D. Watson et al. [WBB13] and "Genes IX" by Benjamin Lewin [Lew07], the structure and function of the DNA are central to genetic studies. sequencing, the process of determining the nucleotide sequence of DNA, has evolved significantly with the advent of Next-Generation Sequencing (NGS). NGS technologies, thoroughly explained in "Next-Generation DNA Sequencing Informatics" by Stuart M. Brown [Bro13] and "Next Generation Sequencing Technologies in Medical Genetics" by C. Alexander Valencia et al. [Val13], have revolutionized genomics by allowing rapid and cost-effective sequencing.

The paired-end sequencing (Figure 2.3), a technique used in NGS, involves sequencing both ends of a DNA fragment to generate high-quality, alignable sequence data. This method allows for the sequencing of both the forward and reverse ends of the fragments, providing two reads per fragment. paired-end sequencing is invaluable for detecting insertions, deletions, and rearrangements, and for improving the genome assembly of complex genomes. In the context of this research, paired-end sequencing can

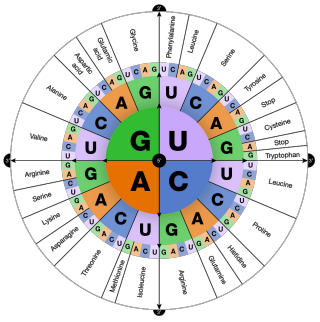

Figure 2.2: **Genetic Code** [Insb] details how genes instruct cells to create proteins, using DNA's four bases.

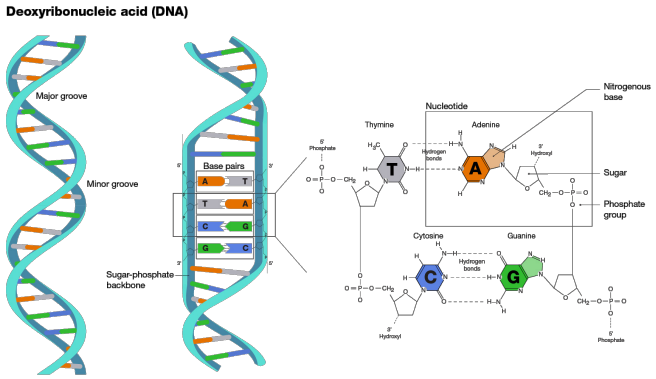

Figure 2.1: **A Diagrammatic Representation of The DNA** double helix structure along with detailed chemical structures of its components [Insa]. On the right is the base pairing between nucleotides: specifically, adenine (A) binds with thymine (T), and cytosine (C) binds with guanine (G), all bound by hydrogen bonds.

enhance the resolution and accuracy of Mycobacterium tuberculosis genome assemblies by providing more information on the spatial relationships between DNA fragments, thus facilitating the identification of genomic variations and structural changes.

2.2 Genome Assembly and Quality Assessment

The NGS is a high-throughput methodology that enables rapid sequencing of the base pairs in DNA or RNA samples. NGS has largely displaced older methods that were based on electrophoresis, as it is faster, more sensitive, and less expensive to operate. Thus, NGS is now used in laboratories ranging from those focused on research on the history of life on Earth to those focused on identifying the exact genetic cause of an individual's disease. The NGS workflow follows several key stages, including Library Preparation (Figure 2.4), Cluster Amplification (Figure 2.5), sequencing (Figure 2.6), and Alignment & Analysis (Figure 2.7). At each stage, the user's understanding of the nature and details of the genetics of an organism grows, from preparing the selected DNA or RNA sample in a usable FASTQ-format [Buf15] to alignment of the NGS reads and analysis of the data. This process involves aligning and merging fragments from a longer DNA sequence, resulting in contigs and scaffolds. The quality of genome assembly is paramount and can

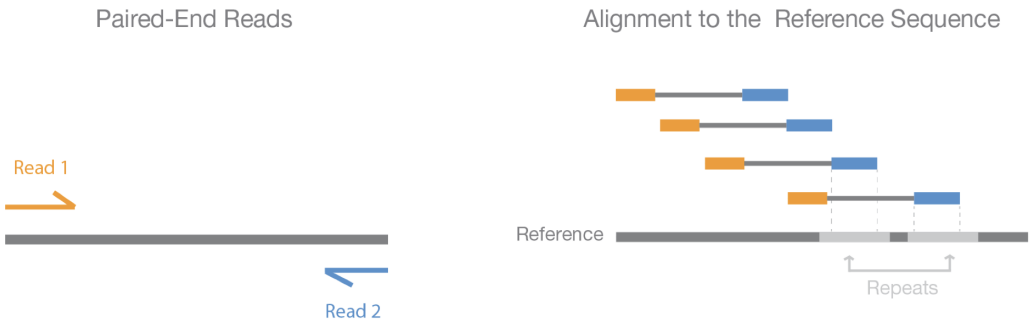

Figure 2.3: **Paired-End Sequencing and Alignment** [Illd] — both ends of a DNA fragment are sequenced. The best advantage of this method is that since the distance between each paired read is known, the alignment algorithms can use this information to map the reads over repetitive regions more accurately. This results in better alignment of reads, especially across difficult-to-sequence, repetitive regions of the genome. Sequences aligned as read pairs allow the detection of indels that are not possible with single-read data.

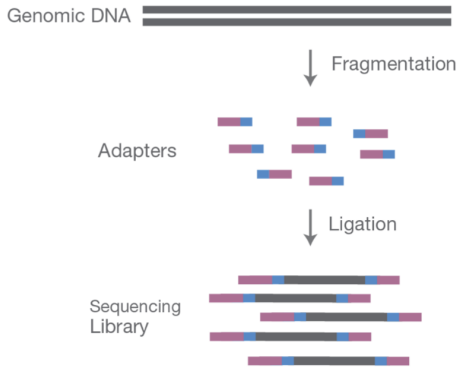

Figure 2.4: **Library Preparation** [Illc] — initial sequencing library prep involves breaking DNA into smaller pieces and adding adapters for fragment amplification and sequencing primer attachment.

be assessed through various assembly metrics, such as N50, N90, L50, L90, total length, GC (%), largest contig, coverage depth, and the number of N's per 100 kbp, which give insights into the size and completeness of the assembled genome.

Every stage of high-throughput sequencing relies on robust and efficient methods to derive the DNA sequence. The first of these steps is the library preparation (Figure 2.4). Genomic DNA must be processed to be ready for sequencing. It is fragmented into small pieces. Adapters are then added to both ends of the fragments. Adapters are short, known DNA sequences. They have several functions. The first is that they form the priming site for the sequencing Process to take place. The adapter also serves as the priming site for the

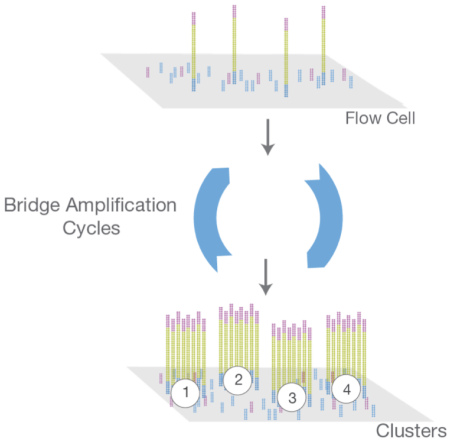

Figure 2.5: **Cluster Amplification** [Illb] — an image of the bridge amplification process, a crucial step in preparing DNA for sequencing on certain high-throughput platforms, such as Illumina Sequencing Systems. DNA fragments with adapters (blue and pink) ligated to their ends are attached to the flow cell surface (gold).

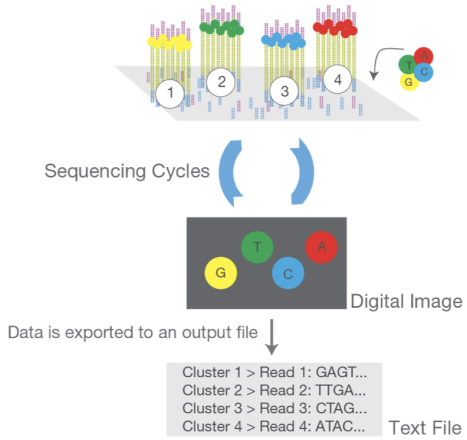

Figure 2.6: **Sequencing** [Ille] — each of the clusters generated from bridge amplification is sequenced one base at a time. A unique fluorescent signal is used to identify each of the four nucleotides (A, T, C, G). As the sequence is built, these signals are captured and translated into a digital image. That digital image is then converted into a text file containing the sequences of the DNA fragments.

amplification (Figure 2.5) of the DNA. The adapter can also contain a barcode, which allows for the pooling and identification of multiple samples within a single sequencing run.

The library, now ready for attachment to a flow cell (Figure 2.5), will undergo clonal amplification on this glass slide. On the surface of the flow cell are lanes that are coated with oligonucleotides that are complementary to the adapters on the fragments. The library is loaded onto the flow cell where the fragments bind to the oligonucleotides. In bridge amplification, the immobilized fragments will bend over and form a bridge and will fragment into oligonucleotides on the other strand. This bridge forms a localized clonal amplification site called a cluster. Within this cluster are many copies of that one DNA.

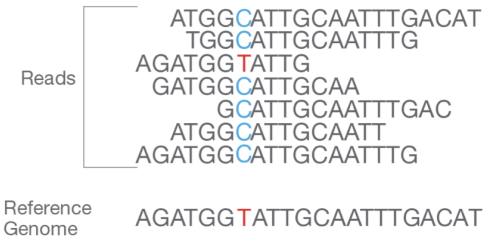

Figure 2.7: **Alignment & Analysis** [Illa] — short DNA sequences (Reads) are matched against a longer 'Reference Genome' sequence. The highlighted sections in the Reads indicate a match, or mismatch, to the Reference Sequence.

The actual sequencing (Figure 2.6) occurs through a series of cycles. Each cycle adds one nucleotide at a time to the DNA strand. The nucleotide, or base, is read by a fluorescent signal; the signal is captured and recorded. In this way, chemical information is translated to digital data (each of the four DNA bases is associated with a unique color). The process cycles, building a sequence read that is complementary to the DNA template in the library.

Resulting sequence reads are computationally aligned to a Reference genome (Figure 2.7); this process identifies where in the genome each read comes from and is critical for variant detection and understanding the structure of the genome. Additionally, reads are assessed for quality; each base call is assigned a quality score, which reflects the confidence of the base call. Ultimately, each analysis reveals variations and mutations, information about genetic predisposition to diseases, and much more.

2.3 Impact of Trimming on Assembly Metrics

The trimming in NGS is a crucial pre-processing step. During this process, low-quality bases and adapter sequences are carefully removed from reads. This significantly affects the quality of the subsequent genome assembly. The effectiveness of different trimming Methods [Ope23], such as adapter trimming, quality trimming, and sliding window trimming, can be evaluated based on their impact on important assembly metrics like N50, GC (%) content, and error rates. These techniques are essential to ensure that only high-quality reads advance to the genome assembly phase, enhancing the overall accuracy and reliability of genomic analysis.

Moreover, the selection of a trimming method can significantly influence downstream analysis and interpretation of NGS data. For instance, quality trimming, which focuses on removing bases with low-quality scores, can markedly decrease false positives in variant calling. Conversely, adapter trimming is vital for avoiding artificial chimeras and ensuring the assembly of genuine genomic sequences. The complexities of these methods underscore the importance of a well-thought-out trimming strategy, customized to the specific needs of each NGS project, to achieve the best results in genomic sequencing and analysis.

2.4 Data Visualization in Genomic Assembly Analysis

Data visualization [CHU07] plays a crucial role in understanding and interpreting the vast amount of data generated by NGS. Visualization tools such as heatmaps, bar plots, and scatter plots can be used to represent various aspects of genome assembly and trimming efficiency. For instance, heatmaps can illustrate the normalized deviation in read quality before and after trimming, while bar plots can show the relationship between trimming methods and key assembly metrics like N50 and number of N's per 100 kbp [Lab22a].

2.5 Conclusion

In summary, NGS and its associated data analysis techniques, including trimming and data visualization, are crucial in assessing genome assembly quality [WW22]. Understanding these concepts, as underlined in the recommended literature, provides vital insights into genomic studies, paving the way for advancements in genetics and beyond.

3 Pipeline Design and Execution

The NGS-pipeline represents a comprehensive solution for NGS data analysis, designed to streamline the process of trimming, genome assembly, and genome assembly quality assessment. By integrating advanced bioinformatics tools such as fastp for trimming, SPAdes for genome assembly, and QUAST for quality evaluation, this pipeline ensures high fidelity in data processing and analysis. The entire codebase of the NGS-pipeline is openly accessible at `https://github.com/shliamin/NGS-pipeline/blob/main/main.nf`, facilitating its adoption and customization for diverse research needs.

3.1 Data Foundation

3.1.1 Input Data

The data for the analysis were sourced from the publicly available sequence read archive (SRA) at the National Center for Biotechnology Information (NCBI). Due to technical constraints and to ensure compatibility with the computing environment, it was decided to directly install SRA-Tools for command-line usage.

The procedure for downloading SRA-formatted data involved the use of the `prefetch` command, followed by conversion to FASTQ-format using `fasterq-dump`, with the `-split-3` option employed for processing Paired-End reads. For instance, to obtain data for Paired genomic DNA sequencing analysis, the following commands were executed:

```
./prefetch SRR26630744
./fasterq-dump --split-3 SRR26630744
```

This process successfully produced the necessary files for the analysis of paired genomic sequencing: `SRR26630744_1.fastq` and `SRR26630744_2.fastq`. This approach requires reads from both ends of a DNA fragment, a technique referred to as paired-end sequencing. Typically, paired-end sequencing involves two files, each containing reads from one end of the fragment. This data was then consistently used for each run of the software pipeline and testing of different methods of trimming, thus producing consistent results within the scientific method of research.

3.1.2 Data Preprocessing

Data preprocessing is a critical step in the pipeline, aimed at improving the quality of the input data for subsequent analyses. The preprocessing includes quality trimming, sliding window trimming, adapter removal, length filtering, and complexity filtering as well as their hybrids with different parameters. These steps are executed using fastp, chosen for its versatility and high performance. The process not only reduces the computational load by removing low-quality sequences but also enhances the reliability of genome assemblies by reducing the number of regions of the genome whose sequence is unknown or uncertain in a given context.

3.2 Software Utilized

This section details the computational tools and software (Table 3.1) utilized in the pipeline. Each tool was selected based on its efficiency, accuracy, and compatibility with the overall design of the pipeline. The versions of the software are specified to ensure reproducibility and consistency of the analysis. The following table summarizes the software tools and their respective versions used:

3.2.1 The fastp For Trimming

The *fastp* is utilized within our pipeline for its efficiency in trimming low-quality bases and removing adapters from raw sequencing reads. The tool's versatility allows for the execu-

Software	Version
Nextflow	23.10.0
fastp	0.23.4
SPAdes	3.15.5
QUAST	5.2.0

Table 3.1: Summary of Software Tools and Versions Used in The Pipeline

tion of a variety of trimming combinations, enhancing the quality of data for downstream analysis. The trimming methods implemented are described below:

The trimming methods: such as *QualityTrim_Q25* and *AdapterTrim_Q25*, employ a systematic approach that combines the trimming method with specific parameters, delineated by underscores (_). Initially, five basic trimming methods (Table 3.2) with averaged parameters are applied, addressing various potential issues in sequencing data:

- **Quality Trimming:** Removes regions with low sequencing quality to improve downstream analysis accuracy.
- **Adapter Trimming:** Excises adapter sequences to prevent interference in read alignment and analysis.
- **Length Filtering:** Retains reads of specified lengths, essential for analyses like assembly.
- **Complexity Filtering:** Eliminates low complexity sequences that may be repetitive or non-informative.
- **Sliding Window Trimming:** Conducts quality trimming within a sliding window, balancing quality control and data retention.

Parameters: follow the trimming method, indicated by a prefix (e.g., *Q25* for a quality threshold of 25, *75* for length filtering at 75 bases).

Hybrids: hybrid methods (Table 3.3), such as *QualityAdapterHybrid_Q25*, integrate two trimming methods, providing a versatile approach to data preprocessing. In the initial run of the pipeline, we explore 16 distinct trimming methods, including a *NoTrimming* option to assess the impact of preprocessing on data quality.

The naming consistency across these combinations plays a critical role in the pipeline's execution, allowing for the traceability of data from input through to the final output. This systematic approach facilitates reproducibility and understanding of the pipeline's workflow, enabling researchers and future users to modify or extend the pipeline with ease.

Demonstration of naming consistency: the logical and transparent naming conventions ensure each data piece can be followed through the pipeline, maintaining the integrity of the workflow and simplifying the reproduction of results.

Figure 3.1: **A Schematic Representation of The Data Flow Through The Trimming, Assembling and Evaluating Stages of The Software Pipeline**. Each trimming method is systematically named to reflect the method and parameters used, ensuring traceability and clarity of the data processing steps.

The Figure 3.1 illustrates the systematic approach to trimming, showcasing the various combinations of methods and their corresponding parameters. This naming consistency allows for each piece of data to be traced through the pipeline, ensuring that the process from initial input to final output, including all intermediate steps, is clear and reproducible.

The trimming methods with their parameters and expected outcomes: in the initial run of the pipeline, various trimming methods (Table 3.2 and Table 3.3) are employed to ensure the highest quality of the sequence data. The table below (Table 3.2) summarizes the 5 basic trimming methods, their descriptions, and the expected outcomes upon processing:

Trimming Method	**Description**	**Expected Result**
NoTrimming	No trimming applied.	Unaltered data
QualityTrim_Q25	Trims low-quality bases below Q25.	Higher quality
AdapterTrim_Q25	Removes adapter sequences.	No adapters
LengthFilter_75	Excludes reads below 75 bases.	Consistent length
ComplexityFilter	Filters low complexity sequences.	Reduced complexity
SlidingWindow _4nt_q25	Sliding window quality trimming: 4nt window, q25 cutoff	Balanced quality

Table 3.2: Basic Trimming Methods in The Initial Run of The Pipeline

Each method is designed to address specific types of artifacts within the sequencing data. The selection of trimming method is crucial to prepare the dataset for accurate assembly and analysis. The most averaged parameters were chosen for each trimming method.

Trimming Method	**Expected Result**
QualitySlidingHybrid_Q25_4nt_q25	Enhanced quality
QualityAdapterHybrid_Q25	Clean reads
LengthComplexityHybrid_75	Lengthy, non-repetitive
SlidingComplexityHybrid_4nt_q25	Quality, non-repetitive
AdapterSlidingHybrid_Q25_4nt_q25	No adapters, balanced
QualityLengthHybrid_Q25_75	High quality, uniform
QualityComplexityHybrid_Q25	High quality, unique
AdapterLengthHybrid_Q25_75	No adapters, consistent
AdapterComplexityHybrid_Q25	No adapters, unique
LengthSlidingHybrid_75_4nt_Q25	Uniform length, balanced

Table 3.3: Hybrid Trimming Methods in The Initial Run of The Pipeline

The hybrid trimming methods (Table 3.3) implemented in the pipeline are the result of combinations without repetitions of the 5 basic trimming methods. Given 5 basic trimming methods, the number of unique hybrid methods that can be created without

repeating any single method can be calculated using the formula for combinations without repetition, also known as binomial coefficients. This is mathematically represented as:

$$C(n,k) = \frac{n!}{k!(n-k)!}$$

For our case, with $n = 5$ basic methods and $k = 2$ methods per combination, the formula simplifies to:

$$C(5,2) = \frac{5!}{2!(5-2)!} = \frac{5 \times 4}{2 \times 1} = 10$$

This combinatorial approach yields exactly 10 unique hybrid methods, which are then utilized in the first run of the software pipeline to ensure a comprehensive exploration of trimming effects. That is the reason for choosing exactly 16 trimming methods (10+5+1), including the *NoTrimming* Method, for the first run of the software pipeline.

The fastp_process: within the pipeline, the *'fastp_process'* is designated to perform sequence trimming and quality control. This process is initiated with a unique label and dynamically assigned tags based on trimming parameters and input file names. The input section specifies a tuple consisting of trimming parameters and associated read files. The output is similarly structured, producing a tuple that includes the original trimming parameters and paths to the trimmed FASTQ-format files. It takes paired-end sequencing reads and the trimming parameters as input and generates trimmed sequence files based on the specified parameters. The process dynamically constructs the fastp command based on the trimming method provided, applying the relevant trimming options.

Below is the code to the *fastp_process*, which is divided into 2 parts (the code snippet 3.1 and the code snippet 3.2) for ease of reading, in which only essential parts of it are indicated, the rest of the points are commented out for brevity.

```
process fastp_process {
    label 'fastp'
    tag "${trimParams}_${reads[0].getName()}"

    input:
        tuple val(trimParams), file(reads)

    output:
        tuple val(trimParams), \
```

```
10          path("${reads[0].simpleName}_${trimParams}.fastq"), \
11          path("${reads[1].simpleName}_${trimParams}.fastq")
12
13      script:
14          String trimParamsStr = trimParams.toString().trim()
15          def fastpCmd = "fastp --in1 ${reads[0]} --in2 ${reads[1]}"
16          String outputFileName1 = "${reads[0].simpleName}_${trimParams}.fastq"
17          String outputFileName2 = "${reads[1].simpleName}_${trimParams}.fastq"
18          fastpCmd += " --out1 ${outputFileName1} --out2 ${outputFileName2}"
19          String htmlReport = "${reads[0].simpleName}_${trimParams}.html"
20          String jsonReport = "${reads[0].simpleName}_${trimParams}.json"
21          fastpCmd += " --html ${htmlReport} --json ${jsonReport}"
22
23          // Representation of dynamic trimming parameters handling (omitted for brevity)
24          // def trimActionMap = [...]
25          // trimActionMap.each { key, action -> ... }
26
27          """
28          set -euo pipefail
29          if [ ${reads.size()} -ne 2 ]; then
30              echo "Error: Expected two reads files, but got ${reads.size()}"
31              exit 1
32          fi
33          ${fastpCmd}
34          """
35  }
```

code snippet 3.1: Comprehensive Overview of the "fastp_process" in the NGS Pipeline

This code snippet 3.1 highlights the input and output file handling, command construction, and application of trimming parameters as part of the *fastp* command execution within the pipeline. The iterative structure ensures that the appropriate action is taken for each set of trimming parameters. The robustness of the script is evidenced by error handling that checks for the expected pair of read files, ensuring data integrity. Upon successful execution, the process concludes with an informative message indicating the completion of the ‘fastp‘ operation for the given parameters.

The core of the ‘fastp_process‘ lies in its script section (the code snippet 3.2), which constructs the ‘fastp‘ command with parameters tailored to the trimming requirements. Here, the trimming parameters are parsed and translated into ‘fastp‘ command options. A mapping structure, ‘trimActionMap‘, is defined to convert symbolic trimming names into their respective ‘fastp‘ command-line arguments. This approach ensures that each trimming method is applied correctly based on the input data characteristics.

```
1  def parts = trimParamsStr.tokenize('_')
2
3      def trimActionMap = [
4      QualityTrim: { String q ->
5        "--qualified_quality_phred ${q.replace('Q', '')}"
6      },
7      AdapterTrim: { String q ->
8        "--detect_adapter_for_pe --qualified_quality_phred " +
9        "${q.replace('Q', '')}"
10     },
```

```
11
12      // The rest of the trimming methods that are omitted for brevity
13
14      NoTrimming: { _ ->
15        "--disable_adapter_trimming --disable_quality_filtering " +
16        "--disable_length_filtering"
17      }
18      ]
19
20      trimActionMap.each { key, action ->
21      if (trimParamsStr.startsWith(key)) {
22          fastpCmd += " " + action(parts.drop(1))
23          return
24        }
25      }
```

code snippet 3.2: Representation of Dynamic Trimming Parameters Handling

For example, based on the trimming method *"QualityTrim_Q25"* a fastp command will be dynamically created (the code snippet 3.2) with the parameter "-qualified_quality_phred" and the quality value 25 will be automatically added (the numbers are also dynamically set) means phred quality >=Q25 is qualified. In this example, a quality value of 25 corresponds to an error probability of 1 in 100 (i.e., 99% accuracy), which is a fairly high threshold for many bioinformatics and genomics applications.

Different combinations (Table 3.4) of sequencing quality parameters (Phred-Score Q) such as Q15, Q20, Q25, and Q30 are presented within this work. These parameters are used to provide varying degrees of rigor in trimming reads, allowing the data cleaning process to be customized for specific study purposes. The relationship between the Phred Score Q and sequencing accuracy is given by the formula:

$$Q = -10 \log_{10}(p)$$

where p is the probability of an error in base calling. The accuracy (%), then, is calculated as:

$$\text{Accuracy } (\%) = (1 - 10^{-\frac{Q}{10}}) \times 100$$

This formula demonstrates how the Phred Score is used to quantify the quality of DNA sequencing, correlating higher scores with higher accuracy (Table 3.4).

Phred-Score Q	Accurancy (%)
Q15	99.97%
Q20	99.99%
Q25	99.999%
Q30	99.9999%

Table 3.4: Correlation between Phred-Score Q and Percentage Accuracy

3.2.2 The SPAdes For Genome Assembly

For the genome assembly stage, *SPAdes* (St. Petersburg genome assembler) was chosen for its versatility and effectiveness in assembling high-quality genomes from NGS data coming from the preceding *fastp_process* (the code snippet 3.1). In our project, *SPAdes* was employed to construct high-quality genomic assemblies from the preprocessed reads. The assembler's advanced algorithms enable the reconstruction of genomic sequences with high accuracy, making it an indispensable tool for our software pipeline.

The spades_process: the code snippet 3.3 represents the *spades_process* within the Nextflow pipeline, aimed at conducting genome assembly utilizing the SPAdes software. This process is integral to transforming sequencing reads into assembled genomes, a crucial step in understanding the genomic architecture of the organism under study.

```
process spades_process {
  label 'spades'
  tag "${trimParams}"

  input:
    tuple val(trimParams), path(read1), path(read2)

  output:
    path "scaffolds_${trimParams}.fasta"

  cpus 4
  memory '8 GB'

  script:
    """
    set -euo pipefail
    spades.py --isolate -1 ${read1} -2 ${read2} -o output_${trimParams}
    mv output_${trimParams}/scaffolds.fasta scaffolds_${trimParams}.fasta
    """
}
```

code snippet 3.3: SPAdes Genome Assembly Process in Nextflow

Key components of the code snippet 3.3:

- **Label and Tag:** The process is labeled 'spades' and tagged with trimming Parameters, facilitating resource allocation and identification of the process's tasks.
- **Input:** Accepts a tuple comprising trimming parameters and paths to paired-end read files, signifying the raw data for genome assembly.
- **Output:** Produces a FASTA file named after the trimming parameters, containing the assembled scaffolds, which are the result of the SPAdes assembly process.
- **Resources:** Allocated with 4 CPUs and 8 GB of memory, ensuring sufficient computational resources for the assembly task. (This parameter can be adjusted to suit the needs and power of the computer individually. This demonstrates the parameters used in this study.)
- **Script:** Executes the SPAdes command with the provided reads, outputting the genome assembly into a designated folder, subsequently moving the scaffolds file to a specified path. The use of `set -euo pipefail` parameter ensures that the script will stop if errors occur, which increases the reliability of the process execution.

This step (in the code snippet 3.3) is crucial in the software pipeline, transforming trimmed sequencing reads into a structured genome, facilitating subsequent genomic analysis.

3.2.3 The QUAST For Quality Assessment

Upon completion of the genome assembly, *QUAST* (Quality Assessment Tool for Genome Assemblies) was utilized to evaluate the quality of the assembled sequences. *QUAST* provides comprehensive quality assembly metrics, including contiguity, coverage, and correctness of the assemblies. By applying *QUAST*, we were able to assess the assembly quality quantitatively, allowing for the identification and correction of potential errors in the code snippet 3.3 assembled genomes. This quality assessment step is crucial for ensuring that the final genomic sequences are accurate and suitable for downstream analyses.

The quast_process: described below (the code snippet 3.4), is an important component of the software pipeline designed to evaluate genomic assemblies derived from the previous *spades_process* using QUAST. This process facilitates genome assembly quality assessment by providing critical insight into the integrity and completeness of assembled genomes. This creates reports on each genome collected here, which we then analyze. Different file extensions are available to choose from, e.g. TSV, which we use in further analysis.

```
process quast_process {

  input:
    path genomes

  output:
    path "reports"

  script:
    """
    mkdir -p reports
    set -euo pipefail
    for genome in ${genomes}
    do
      genome_name=\$(basename \$genome .fasta)
      quast.py -o reports/output_\${genome_name} \${genome}
      mv reports/output_\${genome_name}/report.html \
        reports/report_\${genome_name}.html
    done
    """
}
```

code snippet 3.4: QUAST Quality Assessment Process in Nextflow

Key components of the code snippet 3.4:

- **Input:** It accepts a path to genome assemblies, serving as the input for the quality assessment procedure.
- **Output:** Generates a directory named "reports" which will contain HTML reports for each analyzed genome, summarizing the quality assessment results.
- **Script:** This segment of the process encapsulates the commands for executing QUAST on each genome assembly, highlighting the iterative approach to generate comprehensive quality reports. The use of a loop to process multiple genomes and the subsequent organization of reports demonstrate the process's capability to handle batch assessments, making it a scalable solution for quality analysis.

Overall, the *quast_process* (in the code snippet 3.4) represents an important step towards ensuring the reliability of genomic assemblies by offering a systematic approach to quality control in genomic research.

3.3 Pipeline Implementation

3.3.1 Initial Setup in Nextflow

The provided code snippet 3.5 is a crucial component of a Nextflow pipeline designed for processing NGS data. This segment outlines the initial setup for data preprocessing, specifically focusing on the diversity of trimming methods applied before genome assembly and quality assessment.

```
// Define input and output directory parameters
params.input_dir = 'raw_data'

// Define a list of different trimming combinations for sequence processing
def trimmingCombinations = [
  "QualityTrim_Q25",
  "AdapterTrim_Q25",
  "LengthFilter_75",
  "ComplexityFilter",
  "SlidingWindow_4nt_q25",
  "QualitySlidingHybrid_Q25_4nt_q25", // QualityTrim_Q25 + SlidingWindow_4nt_q25
  // Other trimming combinations
  "NoTrimming"
]

/* Create a channel with trimming combinations
  and pair them with file pairs from the input directory
*/
Channel
  .from(trimmingCombinations)
  .combine(Channel.fromFilePairs("${params.input_dir}/*_{1,2}.fastq"))
  .map { trimParams, sampleId, files ->
    println("Trim Params: ${trimParams}, Files: ${files}")
    tuple(trimParams, files)
  }
  .set { trimmingChannel }
```

code snippet 3.5: Initial Setup and Trimming Strategy Definition in Nextflow

Key components of the code snippet 3.5:

- **Parameter Definition:** The script begins by setting the input directory parameter, which specifies the location of raw sequencing data.

- **Trimming Combinations:** A comprehensive list of trimming methods is defined, encompassing a variety of approaches to clean and prepare the data for subsequent analysis stages. This list demonstrates the pipeline's flexibility in handling different data quality concerns.
- **Channel Creation:** The script dynamically creates a channel that pairs trimming combinations with sequencing file pairs, facilitating the parallel processing of data with various trimming settings.

This script exemplifies a structured and efficient approach to NGS data preprocessing, showcasing Nextflow's capability to orchestrate complex bioinformatics workflows with ease.

3.3.2 Workflow Execution

```
workflow {
  fastp_process(trimmingChannel)
    .groupTuple()
    .set { fastpCollectChannel }

  spades_process(fastpCollectChannel)
    .collect()
    .set { quastInputChannel }

  quast_process(quastInputChannel)
}
```

code snippet 3.6: The Workflow Section

The workflow section (in the code snippet 3.6) illustrates the sequential execution of processes: *'fastp_process'* for read trimming, *'spades_process'* for genome assembly, and *'quast_process'* for assembly quality assessment. This structure underscores the pipeline's cohesive approach to NGS data analysis.

3.3.3 Trimming Strategies

The trimming strategies refer to a predetermined list of trimming methods employed during individual runs of our software pipeline to identify the optimal approach for data preprocessing. These strategies are crucial for enhancing data quality by removing low-quality bases, adapter sequences, and other undesirable elements from raw sequencing

data. The selection of a specific trimming strategy is detailed in the next chapter 4. Within the context of this study, three distinct trimming strategies were implemented, corresponding to three separate executions of the pipeline. Each strategy was designed to explore a range of trimming parameters and their impact on the subsequent analysis, thereby facilitating an informed decision on the most effective preprocessing steps for our dataset.

Three trimming strategies (Table 3.5) were devised to evaluate the performance of various trimming parameters systematically. The strategies encompass a broad spectrum of trimming options, including quality threshold adjustments, adapter trimming, length filtering, and complexity filtering. The intention behind these strategies is to ascertain the combination that yields the highest quality of processed reads, which is pivotal for the reliability of downstream analyses such as genome assembly or variant calling. The results section will detail the effects of using each trimming strategy on the dataset. It will compare their effectiveness using metrics like N50, N90, and the number of N's per 100 kbp among others. Therefore, **48** genomes with different data trimming parameters (Table 3.5) were assembled and analyzed as part of this work.

Table 3.5: Comprehensive Trimming Strategies Employed Across Three Pipeline Runs

The 1st Trimming Strategy	**The 2nd Trimming Strategy**	**The 3rd Trimming Strategy**
QualityTrim_Q25	SlidingWindow_4nt_q15	SlidingWindow_3nt_q30
AdapterTrim_Q25	SlidingWindow_4nt_q20	SlidingWindow_4nt_q30
LengthFilter_75	SlidingWindow_4nt_q25	SlidingWindow_4nt_q29
ComplexityFilter	SlidingWindow_4nt_q30	SlidingWindow_5nt_q30
SlidingWindow_4nt_q25	SlidingWindow_7nt_q15	LengthFilter_30
QualitySlidingHybrid_Q25_4nt_q25	SlidingWindow_7nt_q20	LengthFilter_35
QualityAdapterHybrid_Q25	SlidingWindow_7nt_q25	LengthFilter_40
LengthComplexityHybrid_75	SlidingWindow_7nt_q30	LengthFilter_45
SlidingComplexityHybrid_4nt_q25	SlidingWindow_10nt_q15	LengthFilter_50
AdapterSlidingHybrid_Q25_4nt_q25	SlidingWindow_10nt_q20	LengthFilter_55
QualityLengthHybrid_Q25_75	SlidingWindow_10nt_q25	
QualityComplexityHybrid_Q25	SlidingWindow_10nt_q30	
AdapterLengthHybrid_Q25_75	SlidingWindow_20nt_q15	
AdapterComplexityHybrid_Q25	SlidingWindow_20nt_q20	
LengthSlidingHybrid_75_4nt_Q25	SlidingWindow_20nt_q25	
NoTrimming	SlidingWindow_20nt_q30	
	LengthFilter_50	
	LengthFilter_75	
	LengthFilter_100	
	LengthFilter_150	
	LengthFilter_200	
	LengthFilter_250	

4 Results

4.1 The First Trimming Strategy

The initial phase of our genome assembly investigation centered on applying a pipeline trimming strategy, herein referred to as the "First Trimming Strategy." (Table 4.1). This approach was designed to set a standard against which we could measure the effectiveness of subsequent trimming methods. Our primary objective was to assess its impact on crucial genome assembly metrics. The First Trimming Strategy was implemented with conservative parameters aimed at minimal data loss while ensuring the removal of low-quality sequences. This phase was critical for establishing a control scenario, providing us with a benchmark for evaluating the efficiency and precision of our genome assembly process.

Table 4.1: The First Trimming Strategy and Its Impact on Assembly Metrics

#	Trimming Parameters	Total length	GC (%)	Largest Contig	N50	N90	L50	L90	# N's per 100 kbp
0	NoTrimming	4404338	65.49	209132	74608	21687	18	57	9.20
1	QualityTrim_Q25	4347132	65.39	206803	64262	17217	20	68	7.43
2	AdapterTrim_Q25	4347132	65.39	206803	64262	17217	20	68	7.43
3	LengthFilter_75	4371727	65.42	209097	65136	17421	19	64	6.94
4	ComplexityFilter	4370526	65.41	209097	65136	17421	19	64	6.94
5	SlidingWindow_4nt_q25	4338271	65.45	173819	56417	16144	26	81	7.86
6	QualitySlidingHybrid_Q25_4nt_q25	4338271	65.45	173819	56417	16144	26	81	7.86
7	QualityAdapterHybrid_Q25	4347132	65.39	206803	64262	17217	20	68	7.43
8	LengthComplexityHybrid_75	4371727	65.42	209097	65136	17421	19	64	6.94
9	SlidingComplexityHybrid_4nt_q25	4338271	65.45	173819	56417	16144	26	81	7.86
10	AdapterSlidingHybrid_Q25_4nt_q25	4338271	65.45	173819	56417	16144	26	81	7.86
11	QualityLengthHybrid_Q25_75	4348087	65.39	206803	64262	17217	20	69	7.42
12	QualityComplexityHybrid_Q25	4348353	65.39	206803	64262	17387	20	67	7.43
13	AdapterLengthHybrid_Q25_75	4348087	65.39	206803	64262	17217	20	69	7.42
14	AdapterComplexityHybrid_Q25	4348353	65.39	206803	64262	17387	20	67	7.43
15	LengthSlidingHybrid_75_4nt_Q25	4315732	65.40	173819	54049	16002	27	84	9.06

The data in the Table 4.1 results from reports generated by the first run of the pipeline (subsection A.2.1), its extraction with the code snippet A.2 calling the code snippet A.1, and sorting by using the code snippet A.3, described in the subsection A.2.2.

4.1.1 Data Normalization

Subsequently, the task was set to rethink the visualization of the aforementioned data, aiming for a clearer representation (Figure 4.1). This work started with the definition of pipeline values attributed to the **"No Trimming"** parameter, which were defined as a benchmark or reference condition against which all subsequent evaluations would be compared.

4.1.2 Heatmap Analysis of Trimming Impact

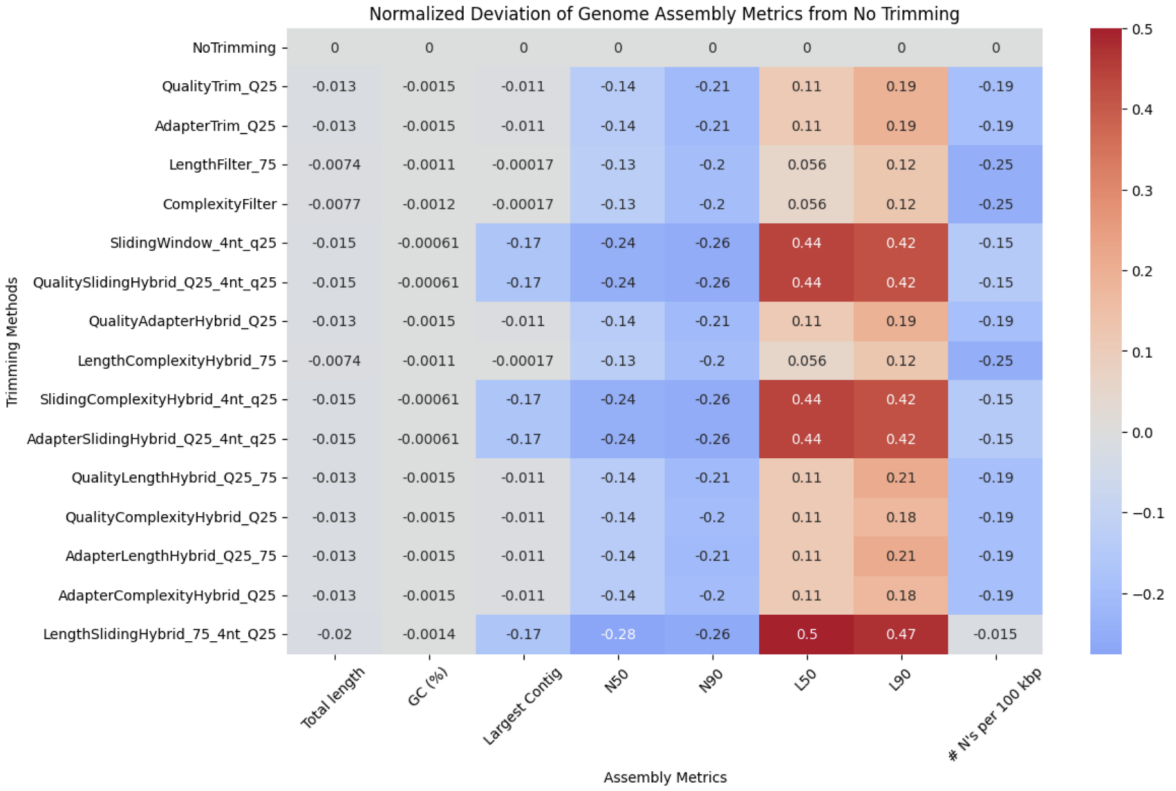

Figure 4.1: **The Normalized Deviation of Genome Assembly Metrics from "No Trimming" — The First Trimming Strategy**. This heatmap illustrates the impact of different trimming methods on various genome assembly metrics. Darker red indicates a greater positive deviation from **"No Trimming"**, while darker blue indicates a greater negative deviation. assembly metrics include total length, GC (%), largest contig, N50, N90, L50, L90, and the number of N's per 100 kbp.

The subsequent step involved normalizing the data: this process is described in the subsection A.2.3, where the use of the code snippet A.4 is explained in detail.

A new DataFrame "norm_deviations" was then generated from a rows_list consisting of dictionaries, with each dictionary representing a row populated with normalized values.

The Figure 4.1 was created as a result of execution the code snippet A.5 described in the subsection A.2.4.

Using the seaborn.heatmap() function, an attempt was made to create a heatmap (Figure 4.1) that eloquently depicts the normalized deviation of genome assembly metrics with respect to the **"No Trimming"** method. In the spatial configuration of the heatmap, rows symbolized the different trimming parameters and columns symbolized the different genome assembly metrics.

The color of the heatmap cells was carefully calibrated to represent the magnitude of normalized deviation variances, with the midpoint anchored to zero. Thus, the gradation from red to blue hues indicated the transition from positive to negative normalized deviation, respectively. This color scheme was chosen to facilitate an intuitive perception of data outliers by providing a sharp visual contrast that highlights the comparative effectiveness of different trimming methods.

4.1.3 Evaluating the Trimming Method's Initial Impact

1. The total length and the GC (%) content are largely unaffected (Figure 4.1) by trimming methods, indicating data consistency and high sequencing quality.
2. The **sliding window trimming**, in all its combinations, leads to a significant decrease in N50 and N90, suggesting an over-aggressive approach that negatively impacts genome assembly.
3. An increase in L50 and L90 across several methods indicates more fragmentation and reduced genome assembly continuity.
4. The **LengthFilter_75** and the **ComplexityFilter** methods are noted for their ability to reduce indefinite nucleotides, improving genome assembly quality.

5. A compromise exists between error reduction and genome assembly continuity, which needs to be balanced for optimal genome assembly outcomes.

6. Further analysis is suggested to assess the compromise between uncertainty and continuity assembly metrics, possibly through additional hierarchical plotting (Figure 4.2) of the trimming methods.

4.1.4 Visual Assessment of Trimming Efficiency

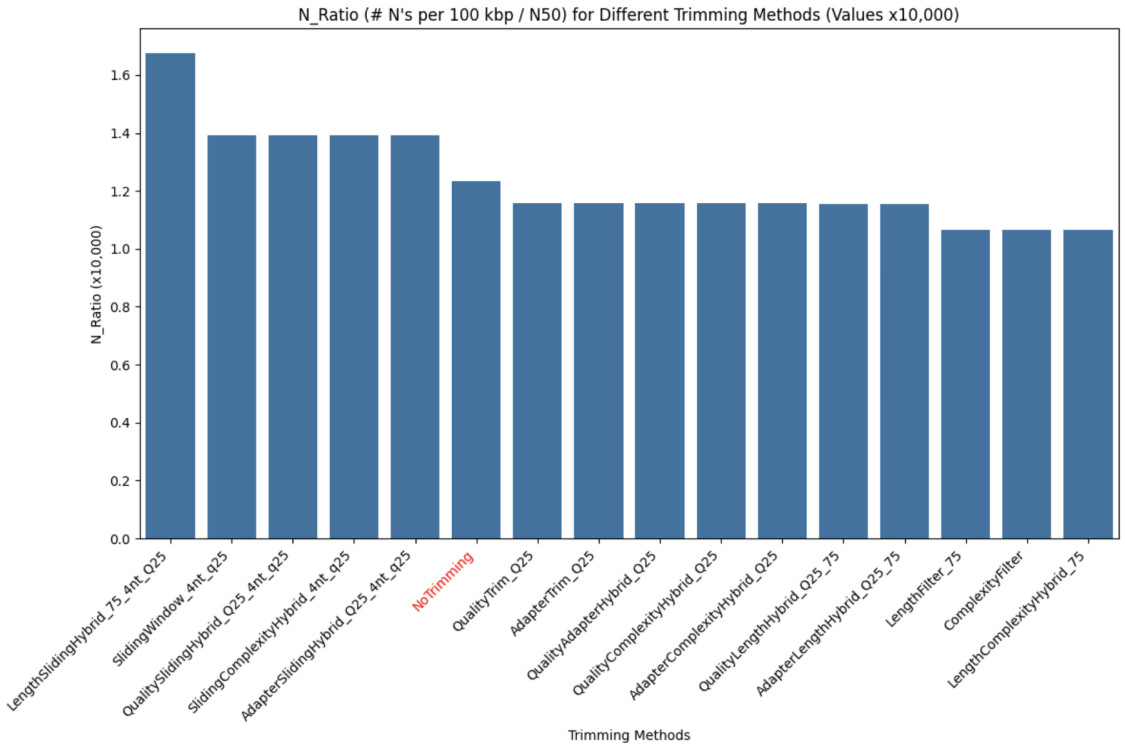

Figure 4.2: **The First Trimming Strategy: The Bar Plots Represent The N Ratios** — The number of N's per 100 kbp to the N50 metric for different trimming methods, with values multiplied by 10,000. The **'No Trimming'** method serves as a reference point. Each bar represents a different trimming method, and the height of the bar indicates the normalized impact of that method on the genome assembly quality. A higher bar denotes a method that results in a higher number of N's per 100 kbp relative to the N50, suggesting a decrease in genome assembly quality.

The Figure 4.2 was obtained by creating the N_ratio, its description is in the subsection A.2.5, where the code snippet A.6 performs its computation, and the subsection A.2.6 describes the visualisation of the N_ratio using the code snippet A.7.

4.1.5 Refining Trimming Parameters Based on Initial Insights

1. The trimming methods generally outperform (Figure 4.2) the **"No Trimming"** by reducing the number of N's per 100 kbp without compromising genome assembly continuity, except for methods involving the **Sliding Window**.
2. The **LengthFilter_75**, the **ComplexityFilter**, and their **hybrid** show the most favorable outcomes in balancing continuity with the reduction of uncertain nucleotides.

To further analyze the impact of trimming methods on genome assembly quality, the following steps are proposed:

1. Re-run the software pipeline with a variety of parameters for the **Sliding Window** and for the **Length Filter** methods using The Second Trimming Strategy (section 4.2).
2. Parameters for the new run should include:
    - 16 different settings for the **Sliding Window** (Table 3.5).
    - 6 different settings for the **Length Filter** (Table 3.5).
3. Compare the outcomes of these runs with the **"No Trimming"** to determine the optimal parameters for each method.

4.2 The Second Trimming Strategy

The data in the Table 4.2 results from reports generated by the second run of the pipeline (subsection A.2.1), its extraction with the code snippet A.2 calling the code snippet A.1, and sorting by using the code snippet A.3, described in the subsection A.2.2.

Table 4.2: The Second Trimming Strategy and Its Impact on Assembly Metrics

#	Trimming Parameters	Total length	GC (%)	Largest Contig	N50	N90	L50	L90	# N's per 100 kbp
0	NoTrimming	4404338	65.49	209132	74608	21687	18	57	9.20
1	SlidingWindow_4nt_q15	4371714	65.47	209097	66370	19392	19	61	7.14
2	SlidingWindow_4nt_q20	4355072	65.47	196013	66370	16891	19	66	6.92
3	SlidingWindow_4nt_q25	4338271	65.45	173819	56417	16144	26	81	7.86
4	SlidingWindow_4nt_q30	4324613	65.42	156069	45172	12240	31	100	4.64
5	SlidingWindow_7nt_q15	4386621	65.47	209097	70975	20667	18	59	11.52
6	SlidingWindow_7nt_q20	4371385	65.48	196013	65416	16973	20	65	11.75
7	SlidingWindow_7nt_q25	4352562	65.46	196013	64262	16735	22	70	7.39
8	SlidingWindow_7nt_q30	4335609	65.44	161559	53760	15315	27	84	10.41
9	SlidingWindow_10nt_q15	4389356	65.48	209097	74525	20679	18	57	11.51
10	SlidingWindow_10nt_q20	4426558	65.49	141613	49888	14028	29	91	20.93
11	SlidingWindow_10nt_q25	4359481	65.47	196013	65247	16891	20	66	7.15
12	SlidingWindow_10nt_q30	4337191	65.45	141533	53760	16002	27	83	8.33
13	SlidingWindow_20nt_q15	4390654	65.47	209097	70975	20679	18	58	9.21
14	SlidingWindow_20nt_q20	4377137	65.47	196013	66370	19418	19	61	6.91
15	SlidingWindow_20nt_q25	4369502	65.47	196013	65416	17217	21	66	11.75
16	SlidingWindow_20nt_q30	4347271	65.46	161570	64262	16677	23	74	10.16
17	LengthFilter_50	4370507	65.41	209097	65136	17421	19	64	6.94
18	LengthFilter_75	4371727	65.42	209097	65136	17421	19	64	6.94
19	LengthFilter_100	4371227	65.42	209075	64713	17421	20	65	6.94
20	LengthFilter_150	4371488	65.42	205948	64262	17416	20	66	6.94
21	LengthFilter_200	4368193	65.42	205900	64262	17416	20	66	7.18
22	LengthFilter_250	4431464	65.42	146203	41434	14197	34	102	16.41

4.2.1 Normalization of Comparative Data

The subsequent step involved normalizing the data: this process is described in the subsection A.2.3, where the use of the code snippet A.4 is explained in detail.

4.2.2 Heatmap Visualization of Refined Data

The Figure 4.3 was created as a result of execution the code snippet A.5 described in the subsection A.2.4.

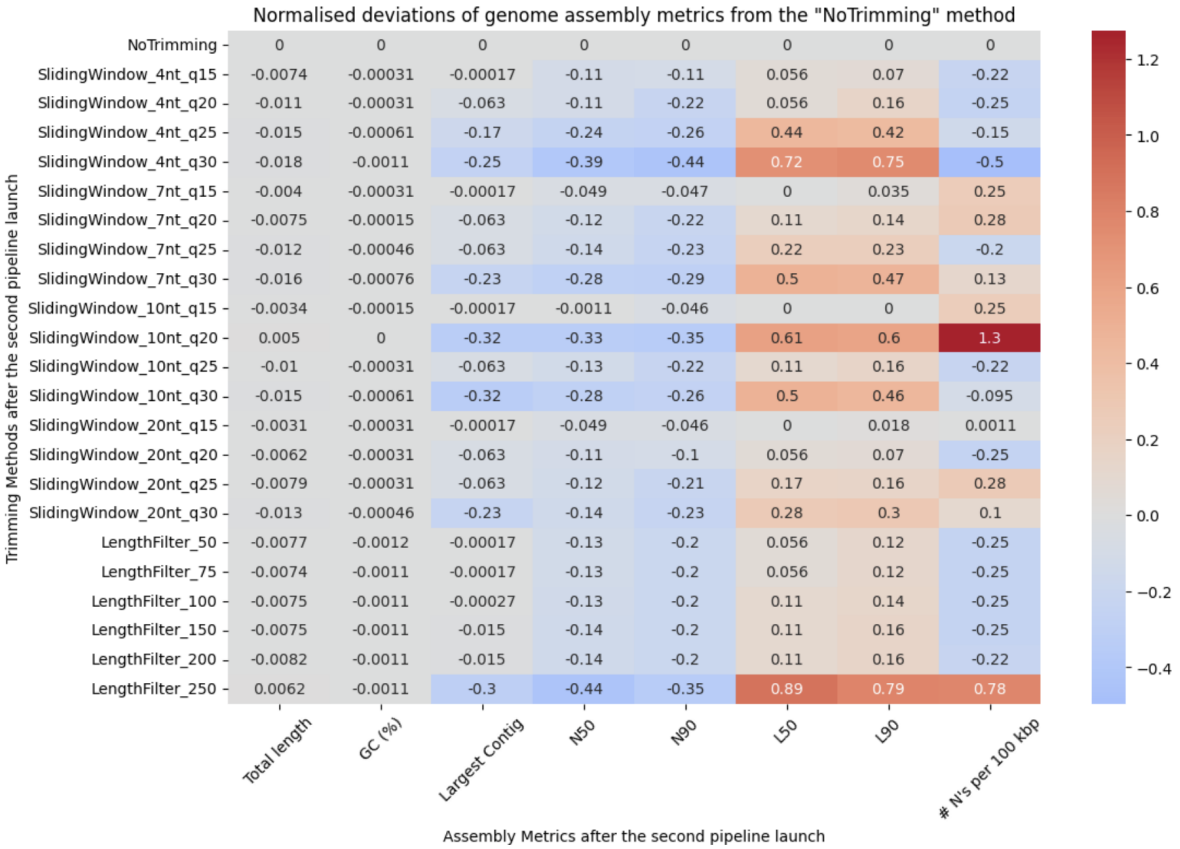

Figure 4.3: **The Normalised Deviation of Genome Assembly Metrics from The "No Trimming" — The Second Trimming Strategy**. This visual representation highlights the impact of different **Sliding Window** and **Length Filter** parameters on assembly metrics such as total length, GC (%), largest contig, N50, N90, L50, L90, and number of N's per 100 kbp. Notably, certain parameters show a marked decrease in N50 and increase in L50, L90, suggesting a compromise between trimming stringency and genome assembly continuity.

4.2.3 Assessment of Refined Trimming Parameters

1. As the number of N's per 100 kbp decreases (Figure 4.3), there is a corresponding decrease in N50 and N90, and an increase in L50 and L90, highlighting a consistent pattern across trimming methods.

2. Some trimming methods, such as the **SlidingWindow_7nt_q15**, lead to an in-

creased number of N's without significantly altering other parameters, indicating a poor trimming performance.

3. Extreme cases, like the **SlidingWindow_10nt_q20** and the **LengthFilter_250**, cause a drastic reduction in genome assembly quality.

To further understand the impact of trimming methods:

1. An additional plot (Figure 4.5) will be created to rank the trimming methods by the ratio of the frequency of N's to the N50 metric, thereby establishing a hierarchy of method effectiveness.

4.2.4 Bar Chart Analysis of N Ratio Improvements

The Figure 4.5 was obtained by creating the N_ratio, its description is in the subsection A.2.5, where the code snippet A.6 performs its computation, and the subsection A.2.6 describes the visualisation of the N_ratio using the code snippet A.7.

```
N E X T F L O W  ~  version 23.10.0
Launching `main.nf` [fervent_galileo] DSL2 - revision: 135392d160
[-        ] process > fastp_process  -
executor >  local (1)
executor >  local (2)
executor >  local (3)
[6f/9a82e6] process > fastp_process (NoTrimming_SRR26630744_1.fastq) [100%] 1 of 1 ✔
[48/a8cb58] process > spades_process (NoTrimming)                    [100%] 1 of 1 ✔
[5a/cc83f2] process > quast_process                                  [100%] 1 of 1 ✔
Completed at: 03-Feb-2024 10:15:48
Duration    : 45m 9s
CPU hours   : 3.0
Succeeded   : 3
```

Figure 4.4: **An Example of An Output of The Nextflow Pipeline Exxecution**, version 23.10.0, showcasing the completion of various bioinformatics processes. The workflow includes processes such as 'fastp_process', 'spades_process', and 'quast_process'. The execution was completed on 03-Feb-2024 at 10:15:48, taking a total duration of 45 minutes and 9 seconds, consuming 3.0 CPU hours, and completing three processes for the single **"NoTrimming"** method.

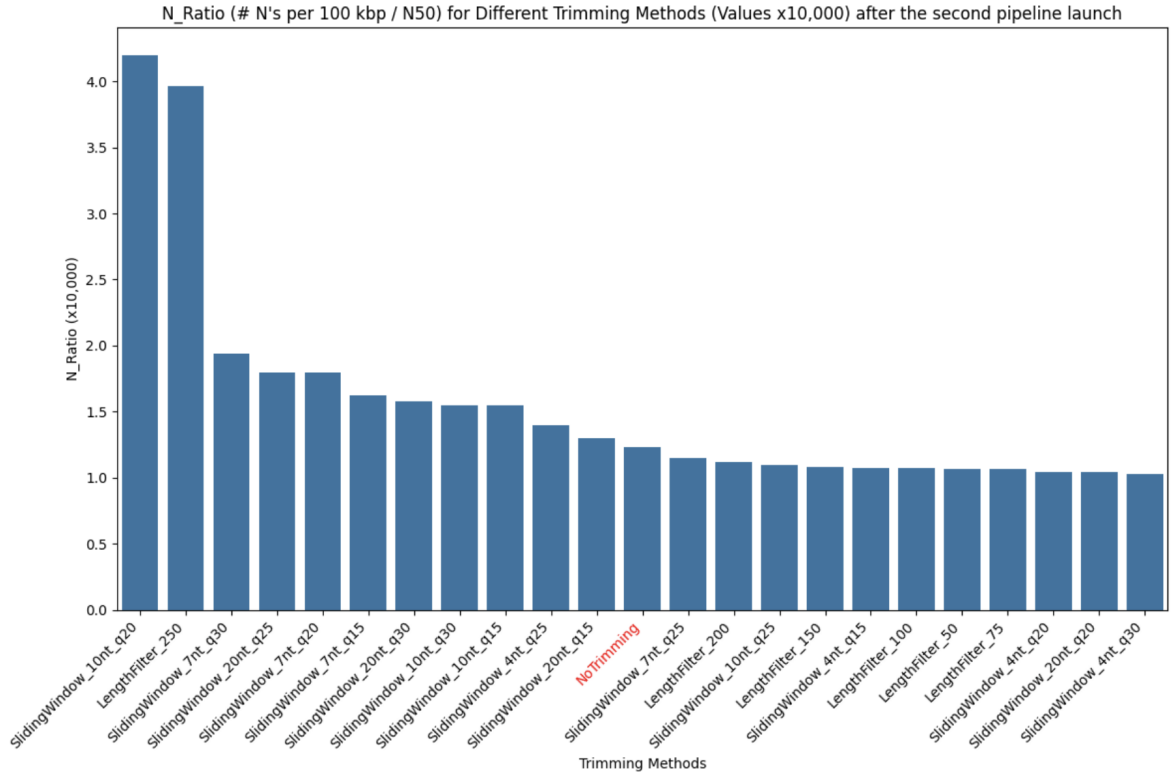

Figure 4.5: **The Second Trimming Strategy: The Bar Plots Showing The N_Ratio**, which is The number of N's per 100 kbp divided by the N50 metric, for different trimming methods following the second pipeline run. The values are multiplied by 10,000 to facilitate comparison. The **'No Trimming'** method serves as a benchmark. A higher bar denotes a method that results in a higher number of N's per 100 kbp relative to the N50, suggesting a decrease in genome assembly quality.

4.2.5 Critical Analysis of Parameter Refinement

1. Certain parameters (Figure 4.5) of the **Sliding Window** method, previously considered suboptimal, have shown improved genome assembly quality upon re-evaluation, even outperforming the previously favored **LengthFilter_75** in some instances.

2. Conversely, some parameters of the initially superior **Length Filter** method, such as the **LengthFilter_250**, may not be advantageous, indicating the significance of parameter selection within each method.

Further Insights and Steps:

1. The importance of the trimming method's parameters has been experimentally validated, influencing the quality of genome assembly.
2. The iterative use of the pipeline has refined our understanding of which trimming methods and parameters yield the best genome assembly quality.
3. A new visualization approach is proposed using scatter plot (Figure 4.6, Figure 4.7) to analyze the variance in genome assembly quality across different parameters within the **Sliding Window** and the **Length Filter** methods.
4. Specifically for the **Length Filter**, the 2D scatter plot (Figure 4.6) will be created to illustrate the relationship between its parameters and the metrics N50 and the number of N's per 100 kbp.

4.2.6 Visualizing Optimal Trimming Outcomes

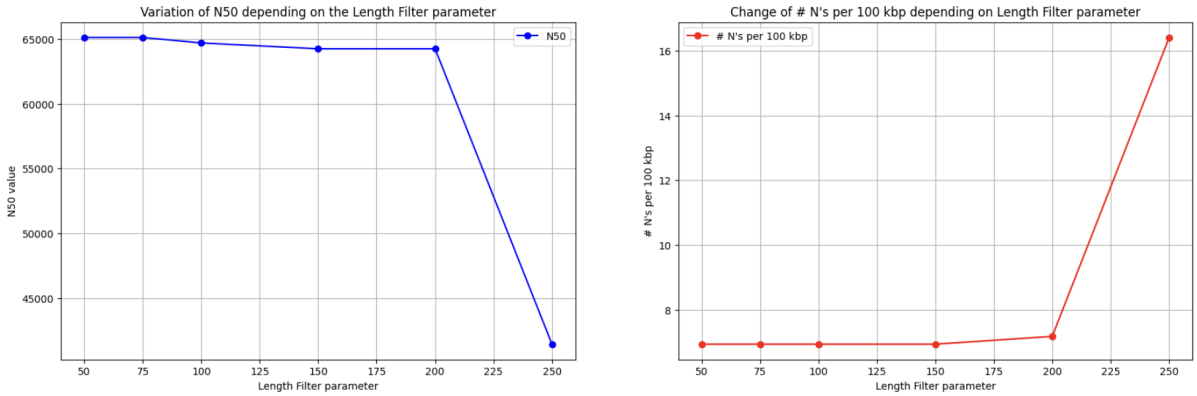

Figure 4.6: **The Dual Scatter Plot Illustrate The Variation of The N50 Metric (Left) and The Change in The Number of N's Per 100 kbp (Right) as A Function of The Length Filter Parameter**. The N50 value remains relatively stable across a range of parameters but sharply decreases after a certain threshold. Conversely, the number of N's per 100 kbp stays consistent before dramatically increasing, indicating a critical parameter value beyond which the **Length Filter** negatively impacts the genome assembly.

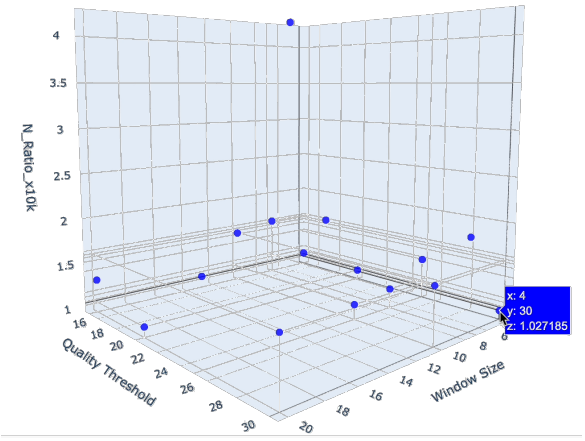

Figure 4.7: **The 3D Scatter Plot Representing The N_Ratio_x10k of The Sliding Window Trimming Method as A Function of Its Parameters**, specifically window size and quality threshold. The plot points are distributed across the three-dimensional space defined by these two parameters and the resulting N_Ratio_x10k, highlighting the complex relationship between trimming method parameters and the impact on genome assembly quality.

The Figure 4.6 was created by using the code snippet A.8 described in the subsection A.2.7 and in the subsection A.2.8. And the Figure 4.7 was created by using the code snippet A.9 described in the subsection A.2.9 and in the subsection A.2.10. The Figure 4.7 depicts just a screenshot of the visual program MetricsExtractor, which interactively allows researches to find the best of the best metrics (subsection A.2.10).

4.2.7 Final Evaluation of Trimming Strategy Efficacy

1. The scatter plots (Figure 4.6, Figure 4.7) reveal an inverse relationship between the N50 metric and the number of N's per 100 kbp, illustrating the impact of the **Length Filter** parameter on genome assembly quality.

2. Parameters in the range of 50 to 75 for the **Length Filter** method are indicated to potentially enhance genome assembly quality.

3. A proposal is made to re-run the pipeline with the **Length Filter** parameters around 50-75 to possibly improve genome assembly quality further.

4. For the **Sliding Window** method, the 3D scatter plot (Figure 4.7) is suggested to analyze the N_Ratio_x10k against its varying parameters, necessitating data filtering to select for the **Sliding Window** parameters exclusively.

5. The interactive 3D scatter plot allows us to examine each data point alongside its corresponding X (window size), Y (mean quality), and Z (N_Ratio_x10k) values, illustrating the trimming method's parameter effects on genome assembly quality.

6. The point with the lowest Z value on the plot indicates the optimal parameters for the **Sliding Window** method that achieves the best balance between the number of N's per 100 kbp and N50, suggesting a superior genome assembly quality.

7. Initiating a third pipeline run with parameters near this optimal point may potentially refine the quality of genome trimming even further.

4.3 The Third Trimming Strategy

Table 4.3: The Third Trimming Strategy and Its Impact on Assembly Metrics

#	Trimming Parameters	Total length	GC (%)	Largest Contig	N50	N90	L50	L90	# N's per 100 kbp
0	SlidingWindow_3nt_q30	4302594	65.40	139540	38864	10691	34	111	1.86
1	SlidingWindow_4nt_q30	4310805	65.42	156069	45172	12240	31	100	4.64
2	SlidingWindow_4nt_q29	4314963	65.43	161279	49758	12487	29	94	6.49
3	SlidingWindow_5nt_q30	4315324	65.43	161302	45552	12261	31	98	6.49
4	LengthFilter_30	4321024	65.41	209097	65136	17421	19	64	6.94
5	LengthFilter_35	4321024	65.41	209097	65136	17421	19	64	6.94
6	LengthFilter_40	4321012	65.41	209097	65136	17421	19	64	6.94
7	LengthFilter_45	4321017	65.41	209097	65136	17421	19	64	6.94
8	LengthFilter_50	4321006	65.41	209097	65136	17421	19	64	6.94
9	LengthFilter_55	4321006	65.41	209097	65136	17421	19	64	6.94

The data in the Table 4.3 results from reports generated by the third run of the pipeline (subsection A.2.1), its extraction with the code snippet A.2 calling the code snippet A.1, and sorting by using the code snippet A.3, described in the subsection A.2.2.

The Figure 4.8 was created by using the code snippet A.8 described in the subsection A.2.7 and in the subsection A.2.8. And the Figure 4.9 was obtained by creating the N_ratio, its description is in the subsection A.2.5, where the code snippet A.6 performs its computation, and the subsection A.2.6 describes the visualisation of the N_ratio using the code snippet A.7.

4.3.1 Comparative Overview of Genome Assembly Metrics

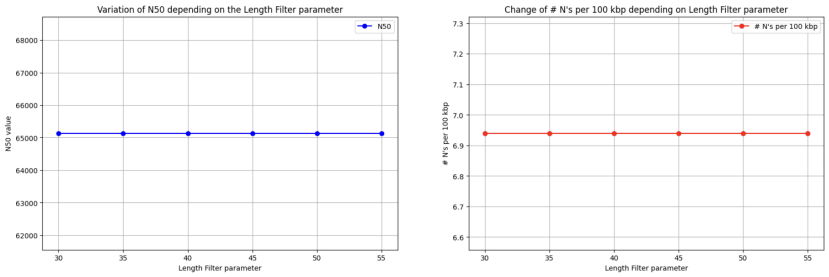

Figure 4.8: **Two scatter plots depicting the variation of the N50 metric and the change in the number of N's per 100 kbp as functions of the Length Filter parameter.** The left plot shows N50 values remaining stable across different parameters, while the right plot illustrates a consistent number of N's per 100 kbp, independent of the **Length Filter** parameter changes. These trends suggest that within the examined parameter range, the **Length Filter** has a negligible effect on both the continuity and the gap size of the genome assembly.

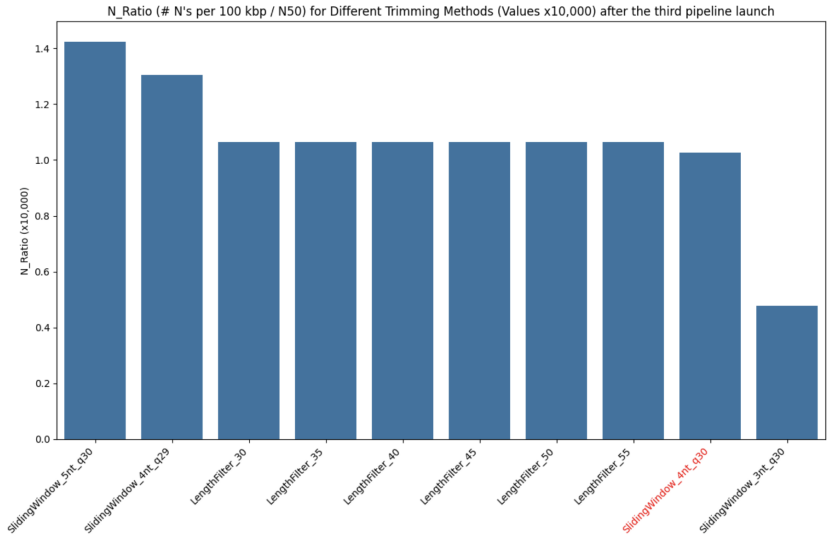

Figure 4.9: **The Bar Plots Showing The N_Ratio for Different Trimming Methods — The Third Trimming Strategy**, multiplied by 10,000. The bars represent various methods, including the **Sliding Window** and the **Length Filter**, with their respective parameter values. The significant decrease in the N_Ratio for certain parameters indicates a more favorable outcome in genome assembly quality.

4.3.2 Conclusive Insights on Trimming Efficacy

1. The analysis of the graph (Figure 4.8) indicates that modifying the trimming method parameters to values near their previously identified optimum does not yield improvement. Consequently, the initial hypothesis was not corroborated by the results.

2. Figure 4.9: our hypothesis was confirmed, leading to the discovery of an improved data trimming method using the **SlidingWindow_3nt_q30** parameters.

4.4 A Comparative Analysis

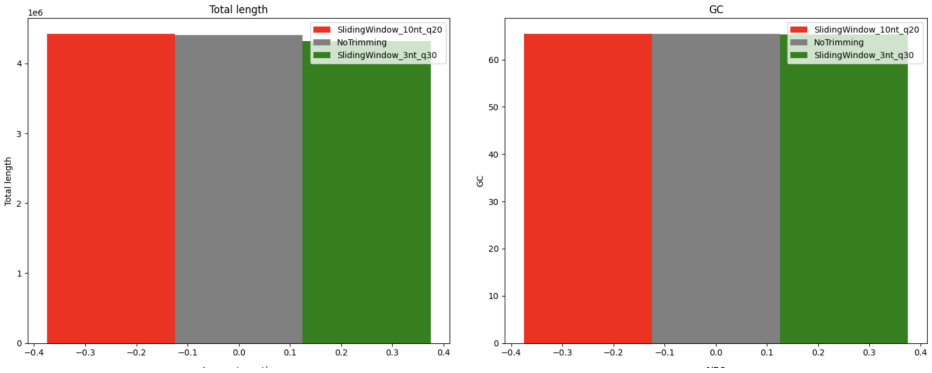

Figure 4.10: **Total Length and GC (%)** - worst/best

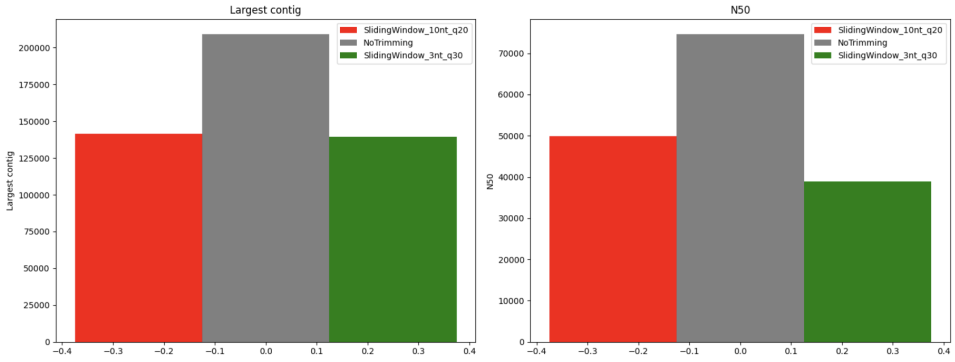

Figure 4.11: **Largest Contig and N50** - worst/best

1. Comparative bar plotss of genome assembly metrics for different trimming methods. Demonstrative comparison of the effects of three trimming Methods on different

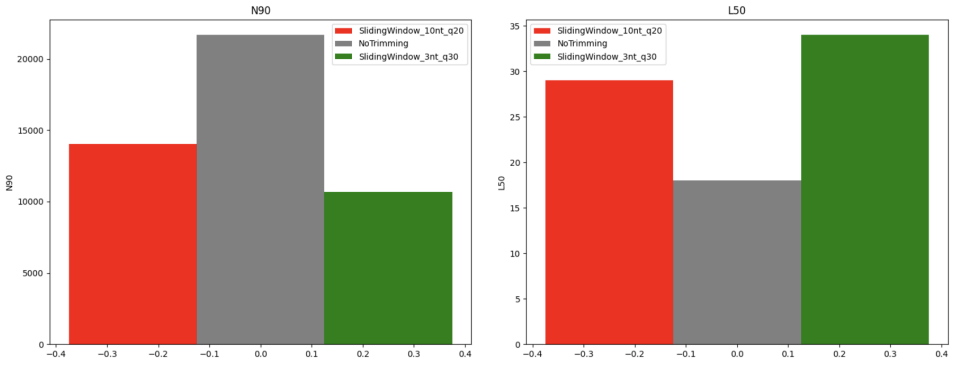

Figure 4.12: **N90 and L50** - worst/best

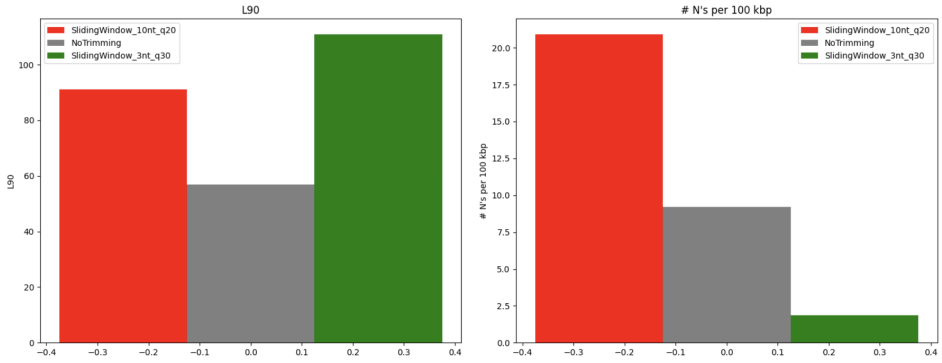

Figure 4.13: **L90 and The Number of N's Per 100 kbp** - worst/best

metrics assessing the quality of genome assembly. The best trimming method found (**SlidingWindow_3nt_q30**) demonstrates the advantage of using the pipeline.

2. Through iterative execution of the pipeline and intermediate data processing, we identified the most and least effective data trimming methods (Figure 4.13), depicted in green and red respectively, against the **"No Trimming"** (gray) scenario.

3. The optimal trimming method, the **SlidingWindow_3nt_q30**, demonstrated the lowest N_Ratio value among all tested, indicating the highest efficiency in reducing the number of N's per 100 kbp relative to N50.

4. The improvement factor for the best trimming method over the no-trimming scenario was **4.95** in terms of number of N's per 100 kbp , showcasing significant enhancement in data quality. The N50 indicator decreased by **1.9** times compared to when trimming was not used.

5 Discussion

Study Overview and Reproducibility. This study embarked upon a comprehensive exploration of various trimming strategies applied to sequenced data, to discern their impact on the quality of genomic assembly. To ensure the reproducibility of our results, meticulous documentation of every procedural step, utilized parameters, and data processing conditions was maintained. This effort culminates in the provision of all data, alongside the LaTeX documentation of this work, which is made available in section A.2.1 of the mentioned repository, facilitating future research endeavors.

Results Interpretation and Novel Metric Introduction. In section 4, we delved into a detailed interpretation of the findings, shedding light on the merits and limitations inherent to each trimming approach. A pivotal aspect of this study is the introduction of a novel metric, termed the N_value, specifically devised to evaluate the efficacy of different trimming methods. This investigation led to the proposition of three iterative trimming strategies, detailed in sections 4.1, 4.2, and 4.3, through which we identified the most efficacious approach.

Significance of the Software Pipeline. The empirical evidence gathered through this research underscores the significance of the software pipeline developed, affirming its utility as an effective tool in the scientific research arsenal. We encourage fellow researchers in the field to engage with the techniques delineated herein and embark on their quest to refine data-trimming strategies, aiming to enhance the stability and reliability of research outcomes.

Selective Focus and Iterative Trimming. Furthermore, this work facilitated the assembly of approximately 100 different genomic assemblies, albeit only 48 are discussed within this paper. This selective focus has underscored a guiding principle: the application of trimming strategies should be iterative, allowing for a nuanced approach to achieving

optimal results. A thorough analysis is recommended following each strategy application, setting the stage for the subsequent trimming iteration.

Limitations and Computational Efficiency. However, a notable limitation of this investigation stems from its oversight of the computational efficiency of each trimming method, considering the assembly and analysis of each genome, compounded by the reliance on a singular local computer's computational resources (Table 5.1). This aspect presents a fertile ground for future research, aiming to bridge this gap and enhance the efficiency of genomic data processing.

Concluding Remarks and Future Directions. In conclusion, while this study makes significant strides toward understanding and improving trimming strategies for genomic data, it acknowledges the necessity for continued research, particularly in optimizing computational resources. We provide a foundation upon which future studies can build, aiming not only to refine these methods but also to expand the horizons of genomic research. Access to our data and documentation, as specified in section A.2.1, invites the scientific community to further this exploration, contributing to the collective advancement of our understanding in this field.

Table 5.1: Summary of Nextflow Workflow Execution Log

Attribute	**Details**
Workflow Command	`nextflow run main.nf`
Nextflow Version	23.10.0
Execution Date	Feb-03
System	Mac OS X 14.0
Java Runtime	OpenJDK 64-Bit Server VM 17.0.6+10-LTS
CPU Cores	8
Memory	8 GB
Processes	`fastp_process`, `spades_process`, `quast_process`
Task Status	All tasks completed successfully
Peak CPUs	4
Peak Memory	8 GB
Execution Complete	Yes

6 Future Prospects

This study opens several pathways for further investigation into the effects of data trimming on genomic analyses. Highlighting potential areas of research, we propose:

Impact on Structural Variations Future work could explore how trimming influences the detection of genomic structural variations, crucial for understanding genomic dynamics.

Gene Annotation Quality Investigating the effect of trimming on gene annotation and function prediction could refine genomic data interpretation.

Antibiotic Resistance Detection Assessing how trimming adjustments affect identifying antibiotic resistance genes may offer insights into combating antimicrobial resistance.

Comparative Genomic Features Analysis of trimming's role in identifying unique versus conserved genomic regions could enhance evolutionary and comparative genomic studies.

Mutation Detection Accuracy Examining how trimming impacts the precision of mutation detection, especially in complex regions, could improve genetic analyses.

Assembly Efficiency Research could evaluate how different trimming approaches optimize genome assembly, affecting speed and accuracy.

Metagenomic Differentiation Studying trimming's effect on differentiating metagenomic samples could advance microbial genomics and environmental biology.

Each direction promises to deepen our genomic understanding, offering a foundation for innovative tools and methods that enhance the precision and reliability of genomic research.

List of Figures

List of Tables

Listings

Glossary

assembly metrics are quantitative measures used to assess the quality and completeness of a genome assembly. These metrics include but are not limited to, the total length of the assembly, the number of contigs, and the N50 statistic, which reflects the contig length such that 50% of the entire assembly is contained in contigs of this length or longer. 5, 7, 8, 18, 24–26, 29, B, C, R–W

bar plots are a type of graph used to represent and compare discrete data or categorical variables. In genomic studies, bar plots can visualize a variety of data, such as the distribution of different genomic features or the abundance of various taxa in metagenomic samples. 8, 36, U

contig is a term derived from "contiguous," indicating a sequence of DNA that has been assembled from overlapping fragments. Contigs represent regions of a genome that have been successfully reconstructed without any gaps. 4

coverage depth is a measure of how many times a particular nucleotide in the genome is represented in the sequencing data. Higher coverage depth increases the reliability of the assembly and the identification of genetic variants. 5

DNA is a double-helical molecule that carries the genetic instructions used in the growth, development, functioning, and reproduction of all known living organisms. The structure comprises two strands forming a double helix, held together by base pairs: adenine with thymine, and cytosine with guanine. This genetic blueprint is crucial for sequencing as it determines the specific order of nucleotides, the foundation of genetic diversity and heredity. 3–7, 9, 10, Q, Y

FASTQ-format is a widely adopted text-based format used for storing both biological sequences, typically nucleotide sequences, and their corresponding quality scores, with both the sequence letter and quality score being encoded using a single ASCII character for efficiency; this format plays a pivotal role in high-throughput sequencing datasets, including those generated by Next-Generation Sequencing (NGS) technologies, making it an indispensable format for handling and analyzing initial data in this study, thereby contributing significantly to the accuracy and reliability of genomic analysis. 4, 9, 14, Q

GC (%) represents the percentage of guanine (G) and cytosine (C) bases in the genome assembly. This metric is crucial as GC content varies significantly across different organisms and can influence sequencing and assembly processes. 5, 7, 24, 25, 29, 36, B–D

genome is the complete set of genetic material, including all the genes and non-coding sequences, contained within a DNA. It serves as the instruction manual for the growth, development, functioning, and reproduction of that organism. Genomes vary in size and complexity among different species and are responsible for the genetic traits and characteristics of individuals within those species. 1–3, 5, 7, 9, 10, 17–19, 22, 24, 25, 34, I, B, U, Y

genome assembly refers to the computational process of reconstructing the complete genomic sequence from fragmented DNA sequences obtained through sequencing. This complex task involves piecing together overlapping sequences to reconstruct the original genome. 1–4, 7–9, 17–23, 25–27, 29–37, I, B–D

heatmap is a graphical representation of data where values in a matrix are depicted as colors. Heatmaps are useful in genomics for visualizing complex datasets, such as gene expression levels or sequence similarities, allowing for easy identification of patterns and outliers. 8, 24, 25, B, T, U

L50 is the number of contigs or scaffolds needed to cover 50% of the genome assembly. It complements the N50 metric by providing a count of the sequences involved. 5, 24, 25, 29, 37, B–D

L90 complements the N90 metric and represents the number of contigs or scaffolds required to cover 90% of the genome assembly. A lower L90 value indicates a more contiguous assembly. 5, 24, 25, 29, 37, B–D

largest contig are the longest contiguous sequences in the assembly. The size of the largest contigs can indicate the assembly's continuity, with longer contigs often signifying a higher-quality assembly. 5, 24, 29, B, C

N50 is a statistical measure that provides the length of the shortest contig or scaffold in the set of largest contigs or scaffolds that together cover at least 50% of the assembly. It is a key indicator of assembly quality, reflecting the contiguity and reliability of the genomic assembly. The N50 metric is especially valuable in gauging the effectiveness of assembly algorithms in producing long, uninterrupted stretches of the genome, which are crucial for accurate gene annotation and the identification of genomic structures. A higher N50 value indicates that a greater portion of the genome is contained within fewer, longer contigs or scaffolds, signifying a more complete and coherent assembly. This measure helps researchers compare assembly performance across different datasets or sequencing technologies, aiming for assemblies that best represent the biological reality of the genome under study. 5, 7, 8, 22, 24–26, 29–37, B–D, U–W

N90 is similar to N50 but more stringent; it represents the length of the contigs or scaffolds in the set of the largest sequences that cumulatively account for 90% of the total assembly length. 5, 22, 24, 25, 29, 37, B–D

NGS refers to a set of high-throughput methodologies that enable rapid sequencing of large segments of DNA base pairs. NGS has revolutionized genomic research by allowing for the sequencing of entire genomes swiftly and economically, surpassing the capabilities of traditional Sanger sequencing. It's particularly relevant in this study for its role in enabling comprehensive analysis of Mycobacterium tuberculosis genomes. 1–4, 7–9, 17, 20, 21, I

normalized deviation is a statistical measure used to compare the variation of a particular metric from a reference or expected value, adjusted for the scale of the data. It is often used to assess the consistency and quality of genomic data across different samples or conditions. 8, 25, U

number of N's per 100 kbp is a critical assembly metric that quantifies the number of undetermined nucleotides ('N') normalized over 100 kilobase pairs of the assembly, providing valuable information about the frequency of gaps within scaffolds or contigs and serving as an indicator of the assembly's completeness and overall quality in genomic analysis. A lower value of N's per 100 kbp indicates a higher quality genome assembly with fewer gaps, suggesting a more continuous and complete representation of the genome. This metric is crucial in de novo genome assembly, where the goal is to reconstruct the genome from short sequencing reads without a reference sequence. It helps researchers evaluate the contiguity and integrity of assembled sequences, guiding improvements in assembly methods and the selection of more accurate and comprehensive genomic datasets for further analysis. 5, 8, 22, 24, 26, 27, 29, 31–35, 37, B–D, U–W

paired-end sequencing is a technique used in next-generation sequencing (NGS) that involves sequencing both ends of a DNA fragment to generate high-quality, alignable sequence data. This method allows for the sequencing of both the forward and reverse ends of the fragments, providing two reads per fragment. Paired-end sequencing is invaluable for detecting insertions, deletions, and rearrangements, and for improving the assembly of complex genomes. In the context of this research, paired-end sequencing can enhance the resolution and accuracy of Mycobacterium tuberculosis genome assemblies by providing more information on the spatial relationships between DNA fragments, thus facilitating the identification of genomic variations and structural changes. 3, 10, 14, Q

read refers to a sequence of nucleotides (the basic units of DNA, represented by the letters A, T, C, and G) that has been obtained from sequencing a DNA molecule. These reads are the raw data produced by DNA sequencing technologies, like Next-Generation Sequencing (NGS) machines, which capture millions of fragments of DNA from a sample, providing a comprehensive snapshot of its genetic composition. The accuracy and depth of coverage of these reads are critical for reliable genomic analysis, determining the resolution at which genetic variations can be identified. Consequently, sophisticated bioinformatics tools are employed to assemble these reads into a coherent sequence, facilitating the identification of genetic markers, mutations, and structural variations within the genome. 1–4, 7–10, 14–18, 21, 22, Q

scaffold represents in genome assembly a higher-level structure that connects several contigs, which are contiguous sequences of DNA, by bridging them with known gaps of unsequenced DNA, often indicated by the letter 'N', thereby offering a more comprehensive representation of the genome's organization and aiding in the approximation of how these sequences are arranged within a chromosome. 2, 4, 18

scatter plot is a graphical method used to display and assess the relationship between two quantitative variables. In the context of genomics, scatter plots can illustrate correlations between different genetic or phenotypic variables, aiding in the identification of potential genetic markers or traits. 8, 32–35, D, X

sequencing is the process of determining the precise order of nucleotides within a DNA molecule. It encompasses any method or technology used to determine the order of the four bases—adenine, guanine, cytosine, and thymine—in a strand of DNA. The advent of rapid DNA sequencing methods has greatly enhanced biological and medical research and discovery. 3–8, 10, 16, 20, 21, 25, Q

total length is the sum of the lengths of all sequences in a genome assembly. It gives an overall size of the assembly, reflecting the extent of the genomic material covered. 5, 24, 25, 29, B, C

trimming is a preprocessing step in genome analysis where low-quality or non-informative sections of DNA sequences are removed. Trimming improves the quality of data for assembly and analysis by eliminating errors and biases, enhancing the precision and dependability of subsequent computational assessments. This process is essential for minimizing the introduction of artefacts and improving the overall integrity of genomic interpretations. 1, 2, 7–16, 18, 20–27, 29–34, 36, 37, I, B–D, R, T–X

[style=dottedlocations,type=acronym,title=Abkürzungsverzeichnis]

Bibliography

[Bro13] Stuart M. Brown. *Next-Generation DNA Sequencing Informatics*. Cold Spring Harbor Laboratory Press,U.S., 2013. ISBN: 978-1621821236.

[Buf15] Vince Buffalo. *Bioinformatics Data Skills: Reproducible and Robust Research With Open Source Tools*. O'Reilly Media, 2015. ISBN: 978-1449367374.

[CHU07] Chun-houh Chen, Wolfgang Karl Haerdle, and Antony Unwin. *Handbook of Data Visualization*. Springer, 2007. ISBN: 978-3540330363.

[JP04] Neil C. Jones and Pavel A. Pevzner. *Introduction to Bioinformatics Algorithms*. MIT Press, 2004. ISBN: 0262101068.

[Lew07] Benjamin Lewin. *Genes IX*. Jones and Bartlett Publishers, Inc, 2007. ISBN: 978-0763752224.

[Pat15] Elaine Patrick. *Next-Generation DNA Sequencing Informatics, Second Edition*. CRC Press, 2015. ISBN: 1621821234.

[RB21] Lavanya Rishishwar and Anirban Bhattacharya. *Next Generation Sequencing and Data Analysis*. Springer, 2021. ISBN: 3030624897.

[SO22] Sewe S.O. et al. *Trimming and Validation of Illumina Short Reads Using Trimmomatic, Trinity Assembly, and Assessment of RNA-Seq Data*. Humana, New York, NY, 2022. ISBN: 978-1-0716-2066-3.

[Val13] C. Alexander Alexander Valencia et al. *Next Generation Sequencing Technologies in Medical Genetics*. Springer, 2013. ISBN: 978-1461490319.

[Wan17] Xinkun Wang. *Next Generation Sequencing Data Analysis*. CRC Press, 2017. ISBN: 1482217880.

[WBB13] James D. Watson, Tania A. Baker, and Stephen P. Bell. *Molecular Biology of the Gene*. Benjamin-Cummings, 2013. ISBN: 978-0321762436.

Online Sources

[Con24a] Matplotlib Contributors. *Matplotlib: Plotting with Python*. GitHub repository. 2024. URL: https://github.com/matplotlib/matplotlib.

[Con24b] NumPy Contributors. *NumPy: The fundamental package for scientific computing with Python*. GitHub repository. 2024. URL: https://github.com/numpy/numpy.

[Con24c] Plotly Contributors. *Plotly: Data Visualization in Python*. GitHub repository. 2024. URL: https://github.com/plotly.

[Con24d] Project Jupyter Contributors. *Jupyter Interactive Notebook*. GitHub repository. 2024. URL: https://github.com/jupyter/notebook.

[Con24e] Seaborn Contributors. *Seaborn: Statistical Data Visualization in Python*. GitHub repository. 2024. URL: https://github.com/mwaskom/seaborn.

[Del13] C. Del Fabbro et al. *An extensive evaluation of read trimming effects on Illumina NGS data analysis*. 2013. DOI: 10.1371/journal.pone.0085024.

[Heu21] Tim H. Heupink et al. *Comprehensive and accurate genetic variant identification from contaminated and low-coverage Mycobacterium tuberculosis whole genome sequencing data*. 2021. DOI: 10.1099/mgen.0.000689.

[JBA18] Kemal J., Sibhat B., and Abraham. *Bovine tuberculosis in eastern Ethiopia: prevalence, risk factors and its public health importance*. 2018. DOI: https://doi.org/10.1186/s12879-018-3628-1.

[Lab22a] A.B. Lab. *QUAST: Quality Assessment Tool for Genome Assemblies*. GitHub repository. 2022. URL: https://github.com/ablab/quast.

[Lab22b] A.B. Lab. *SPAdes: St. Petersburg genome assembler*. GitHub repository. 2022. URL: https://github.com/ablab/spades.

[Lia20] Xingyu Liao et al. *An Efficient Trimming Algorithm based on Multi-Feature Fusion Scoring Model for NGS Data*. 2020.

[Net22] Thidarat Netikul et al. *Whole-genome single nucleotide variant phylogenetic analysis of Mycobacterium tuberculosis Lineage 1 in endemic regions of Asia and Africa*. 2022. DOI: 10.1038/s41598-022-05524-0.

[Ope23] OpenGene. *Fastp: An Ultra-fast All-in-one FASTQ Preprocessor*. GitHub repository. 2023. URL: https://github.com/OpenGene/fastp.

[SSB16] Marc Sturm, Christa Schroeder, and Peter Bauer. *SeqPurge: highly-sensitive adapter trimming for paired-end NGS data*. 2016. DOI: 10.1186/s12859-016-1069-7.

[STG21] Lusayda Sanchez-Corrales, Olga Lucia Tovar-Aguirre, and Narmer Fernando Galeano-Vanegas. *Phylogenomic analysis and Mycobacterium tuberculosis antibiotic resistance prediction by whole-genome sequencing from clinical isolates of Caldas, Colombia*. 2021. DOI: 10.1371/journal.pone.0258402.

[Tea24a] Nextflow Team. *Nextflow: Data-driven computational pipelines*. GitHub repository. 2024. URL: https://github.com/nextflow-io/nextflow.

[Tea24b] Pandas Development Team. *Pandas: Flexible and Powerful Data Analysis / Manipulation Library for Python*. GitHub repository. 2024. URL: https://github.com/pandas-dev/pandas.

[VS17] M. D. Vishwasrao and A. K. Sangaiah. *ESCAPE: Effective Scalable Clustering Approach for Parallel Execution of Continuous Position-Based Queries in Position Monitoring Applications*. 2017. DOI: 10.1109/TSUSC.2017.2690378.

[Wag21] Darlene D. Wagner et al. *Evaluating whole-genome sequencing quality metrics for enteric pathogen outbreaks*. 2021. DOI: 10.7717/peerj.12446. URL: https://peerj.com/articles/12446/.

[WW22] Peng Wang and Fei Wang. *A proposed metric set for evaluation of genome assembly quality*. 2022. DOI: https://doi.org/10.1016/j.tig.2022.10.005.

[Yan19] Shang-Fang Yang et al. *To Trim or Not to Trim: Effects of Read Trimming on the De Novo Genome Assembly of a Widespread East Asian Passerine, the Rufous-Capped Babbler (Cyanoderma ruficeps Blyth)*. 2019. DOI: 10.3390/genes10100737. URL: https://www.mdpi.com/2073-4425/10/10/737.

Image Sources

[Illa] Inc. Illumina. *Alignment and Data Analysis - during data analysis and alignment, the newly identified sequence reads are aligned to a reference genome.* URL: https://emea.illumina.com/content/dam/illumina-marketing/documents/products/illumina_sequencing_introduction.pdf.

[Illb] Inc. Illumina. *Cluster Generation - the library is loaded into a flow cell where fragments are captured on a lawn of surface-bound oligos.* URL: https://emea.illumina.com/content/dam/illumina-marketing/documents/products/illumina_sequencing_introduction.pdf.

[Illc] Inc. Illumina. *LibraryPreparation - prepared by random fragmentation of the DNA or cDNA sample.* URL: https://emea.illumina.com/content/dam/illumina-marketing/documents/products/illumina_sequencing_introduction.pdf.

[Illd] Inc. Illumina. *Paired-End Sequencing and Alignment.* URL: https://emea.illumina.com/content/dam/illumina-marketing/documents/products/illumina_sequencing_introduction.pdf.

[Ille] Inc. Illumina. *Sequencing - the result is highly accurate base-by-base sequencing.* URL: https://emea.illumina.com/content/dam/illumina-marketing/documents/products/illumina_sequencing_introduction.pdf.

[Insa] National Human Genome Research Institute. *A schematic representation of the DNA double helix structure along with detailed chemical structures of its components.* URL: https://www.genome.gov/genetics-glossary/Deoxyribonucleic-Acid.

[Insb] National Human Genome Research Institute. *Genetic-Code - refers to the instructions contained in a gene that tell a cell how to make a specific protein.* URL: https://www.genome.gov/genetics-glossary/Genetic-Code.

A

A.1 Preparing The Development Environment

This diversion, though not directly tied to the subsequent discussion on pipeline progress but concentrated exclusively on software configurations, could be invaluable for other researchers. The setup process was particularly labor-intensive, which justifies the comprehensive explanation provided. *Singularity* (`https://docs.sylabs.io/guides/latest/user-guide/quick_start.html`), though initially developed for Linux, can be run on Windows and Mac systems through the use of virtual machines. For Mac users, as in this case, the preliminary step entails the installation of virtualization software, crafting a Linux disk image, and establishing a virtual machine to facilitate the operation of Linux services and applications in a segregated setting, a step deemed essential for employing Singularity. On macOS, tools like VirtualBox, Vagrant, and Vagrant Manager were effortlessly installed via Homebrew and Cask. The complication arose with the Apple M1 chip, which operates on ARM architecture, diverging from the conventional x86_64 architecture seen in Intel/AMD processors. Considering the progressive development of virtualization support for ARM architecture, considerable effort was dedicated to identifying an appropriate solution. *Parallels Desktop* (`https://www.parallels.com/products/desktop/`) was identified as the most straightforward resolution to this challenge. Following this, all subsequent steps were replicated in a Linux Ubuntu setting:

```
# Download Singularity image
wget https://depot.galaxyproject.org/singularity/sra-tools%3A3.0.9--h9f5acd7_0
# Rename the file due to its unwieldy name
mv sra-tools\:3.0.9--h9f5acd7_0 sratools.sif
# Use prefetch to download the data for the specified run in SRA format
```

```
singularity exec sratools.sif prefetch SRR26630744
# Convert downloaded SRA data to FASTQ format, --split-3 is required for
# paired-end reads
singularity exec sratools.sif fasterq-dump --split-3 SRR26630744
```

Furthermore, a complication was faced with the Go binary file, essential for installing Singularity, due to incompatibility with the architecture of Apple's M1 chip. This led to the inability to employ a version of Go tailored for the ARM64 architecture, necessitating the use of an alternate computer equipped with a Linux Ubuntu system and an Intel processor. An attempt was made to remotely install Singularity using *AnyDesk software* (`https://anydesk.com/en`), following a comprehensive review of the installation guidelines available at `http://sylabs.io`. Regrettably, this method was also found to be overly time-consuming.

Consequently, it was decided to bypass Singularity and directly install SRA tools for command-line usage instead. The compatible version for the computer was located at `https://github.com/ncbi/sra-tools/wiki/01.-Downloading-SRA-Toolkit`. The procedure involves downloading SRA-formatted data using prefetch and converting it to FASTQ-format with fasterq-dump. It's important to highlight the use of the –split-3 option for processing paired-end reads, illustrating the steps necessary for this approach:

```
./prefetch SRR26630744
./fasterq-dump --split-3 SRR26630744
```

This procedure successfully produced the needed for the Paired Genomic sequencing Analysis **SRR26630744_1.fastq** and **SRR26630744_2.fastq** files. This analysis requires reads from both ends of a DNA fragment, a technique referred to as paired-end sequencing. paired-end sequencing usually involves two files, each containing reads from one end of the fragment. Although this solution might seem direct, reaching this stage was a prolonged effort. Moreover, initial attempts to run commands in the Mac terminal encountered obstacles due to security settings, which required additional adjustments to the configuration.

A.2 Technical Annexes and Codes

Beyond developing the main software pipeline, an additional tool, **MetricsExtractor.ipynb**, was created and is available on GitHub at `https://github.com/shliamin/NGS-pipeline/blob/main/MetricsExtractor.ipynb`. This supplementary program is crucial for analyzing and visualizing data produced by the pipeline. By evaluating key assembly metrics from different pipeline executions, MetricsExtractor.ipynb supports a scientific research methodology aimed at optimizing the trimming process of sequenced data.

A.2.1 Generated Data

To facilitate a comprehensive understanding and ensure transparency in the research process, all relevant data generated across various stages of pipeline execution are made available online. These datasets provide insights into the relevant outcomes of each experimental run of the pipeline, allowing for detailed scrutiny and comparison:

- Data from the **first execution** of the software pipeline can be found at `https://github.com/shliamin/NGS-pipeline/tree/main/reports`.
- Results following the **second execution** are accessible at `https://github.com/shliamin/NGS-pipeline/tree/main/reports_2`.
- The **third round** of execution data is available at `https://github.com/shliamin/NGS-pipeline/tree/main/reports_3`.
- Additionally, data from **trials exploring the extremities** of processing parameters are provided at `https://github.com/shliamin/NGS-pipeline/tree/main/TrialsExtremes`.

These resources not only highlight the iterative and exploratory essence of scientific research but also act as a valuable archive for the academic community. They provide a means to analyze, replicate, or further develop the discoveries made in this study, fostering an environment of continuous learning and innovation.

A.2.2 Data Extraction

Here are code snippets from the mentioned program, the use of which is explained in the main body of the thesis. To keep the document readable, only the most relevant parts of the code are given here with their explanations, the whole code can be seen at the GitHub link provided above.

Below is the code snippet A.1 demonstrates the *Metrics Extraction Function* relevant to our study from the data generated by the software pipeline.

```
import pandas as pd

def extract_required_metrics(report_path):
    # Reading the report file into a DataFrame
    df = pd.read_csv(report_path, sep='\t', header=None, names=['Metric', 'Value'])

    metrics = {} # Dictionary to store metrics
    required_metrics = [
        ('Total length', 'Total length'),
        ('GC', 'GC (%)'),
        ('Largest contig', 'Largest Contig'),
        ('N50', 'N50'),
        ('N90', 'N90'),
        ('L50', 'L50'),
        ('L90', 'L90'),
        ('# N\'s per 100 kbp', '# N\'s per 100 kbp')
    ]

    # Extracting required metrics
    for metric_pattern, metric_name in required_metrics:
        value = df[df['Metric'].str.contains(metric_pattern)]['Value'].values
        metrics[metric_name] = value[0] if value.size > 0 else 'N/A'

    return metrics
```

code snippet A.1: Metrics Extraction Function

The core functionality involves reading report files by using the *Metrics Extraction Function* (the code snippet A.1), each containing data in a tab-separated format, and extracting key assembly metrics. The process is summarized as follows (the code snippet A.2):

```
# Extracting and saving data from all report files
all_metrics_data = {}
for report_file in report_files:
    report_path = os.path.join(reports_dir, report_file)
    trimming_param = report_file.replace('report_scaffolds_', '').replace('.tsv', '')
    metrics = extract_required_metrics(report_path)
    if metrics:
        all_metrics_data[trimming_param] = metrics
```

code snippet A.2: Extracting Metrics from Report Files

After extracting the assembly metrics (in the code snippet A.2), the data is organized and sorted according to predefined criteria[1] for better analysis and presentation (in the code snippet A.3):

```
# Sorting the DataFrame according to the specified order
sorted_metrics_df = all_metrics_df.set_index('Trimming Parameters').reindex(sorting_order).reset_index()
```

code snippet A.3: Sorting Extracted Metrics

This is how the process of extraction of TSV files generated with the software pipeline is described. Each new start of the pipeline selects a new sorting order criteria suitable for this stage of the study. Thus, by changing only a small part of the parameters, new data can be obtained steadily.

A.2.3 Data Normalization

This subsection discusses the methodology for normalizing data metrics and their subsequent visualization through a heatmap (the code snippet A.5), facilitating the comparative analysis of genomic assembly metrics.

Normalization (the code snippet A.4) involves adjusting the Data assembly metrics to a common scale without distorting differences in the ranges of values. This process is crucial for comparing assembly metrics across different trimming methods.

```
# Ensure the data is numerical and normalize it
for columns in sorted_metrics_df.columns[1:]:
    sorted_metrics_df[column] = pd.to_numeric(sorted_metrics_df[column], errors='coerce')

# Calculate the base values for comparison
base_values = sorted_metrics_df[sorted_metrics_df['Trimming Parameters'] == 'NoTrimming'].iloc[0, 1:]

# Normalize the data relative to the base values
rows_list = []
for index, row in sorted_metrics_df.iterrows():
    norm_row = {'Trimming Parameters': row['Trimming Parameters']}
    for metric in sorted_metrics_df.columns[1:]:
        if base_values[metric] != 0:
            norm_row[metric] = (row[metric] - base_values[metric]) / base_values[metric]
        else:
            norm_row[metric] = 0
    rows_list.append(norm_row)

norm_deviations = pd.DataFrame(rows_list)
```

code snippet A.4: Data Normalization Process

[1]This criteria varies and is described in the MetricsExtractor.ipynb program on a case-by-case basis

A.2.4 Visualization with Heatmap

The normalized data is visualized using a heatmap (code snippet A.5), which represents the normalized deviation of each metric from the pipeline established by the **"No Trimming"** method.

```
# Visualize the normalized deviations
plt.figure(figsize=(12, 8))
sns.heatmap(norm_deviations.iloc[:, 1:], annot=True, cmap='coolwarm', center=0, yticklabels=norm_deviations['Trimming Parameters'])
plt.title('Normalized Deviation of Genome Assembly Metrics from No Trimming')
plt.xlabel('Assembly Metrics')
plt.ylabel('Trimming Methods')
plt.xticks(rotation=45)
plt.show()
```

code snippet A.5: Heatmap Visualization

The normalization and visualization process (A.2.3 and A.2.4) enables a clear, comparative view of how different trimming methods affect genome assembly metrics, highlighting the effectiveness of each method concerning untrimmed data.

A.2.5 Calculation of The "N Ratio"

This subsection elaborates on the calculation of a specific ratio, referred to as the N Ratio, and its visualization through a bar plots graph to compare the effect of different trimming methods on genomic data.

The N Ratio is calculated (in the code snippet A.6) to establish a relationship between the number of N's per 100 kbp and the N50 assembly metrics. This ratio is then scaled by a factor of 10,000 for better representation.

```
# Calculate N Ratio and add it as a new column
sorted_metrics_df['N_Ratio_x10k'] = (sorted_metrics_df['# N\'s per 100 kbp'] / sorted_metrics_df['N50']) * 10000
```

code snippet A.6: Calculating of the "N Ratio"

A.2.6 Visualization of The "N Ratio" With A Bar Graph

The calculated N Ratios are visualized using bar plots (the code snippet A.7), highlighting the comparison across different trimming methods with a specific focus on the **'NoTrimming'** method, marked distinctly for easy identification.

```
# Visualization of N Ratio across trimming methods
plt.figure(figsize=(12, 8))
barplot = sns.barplot(x='Trimming Parameters', y='N_Ratio_x10k', data=sorted_df)
plt.title('N_Ratio (# N\'s per 100 kbp / N50) for Different Trimming Methods (Values x10,000)')
plt.xlabel('Trimming Methods')
plt.ylabel('N_Ratio (x10,000)')

# Highlight 'NoTrimming' method in red
for label in barplot.get_xticklabels():
    if label.get_text() == 'NoTrimming':
        label.set_color('red')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

code snippet A.7: Visualization of the "N Ratio"

The methodology described in subsections A.2.5 and A.2.6 facilitates an insightful comparison of the impact of various trimming methods on the genomic assembly metrics, specifically through the lens of the N Ratio.

A.2.7 Data Filtering: The "Length Filter" Trimming Method

Data corresponding to the **"Length Filter"** trimming method were isolated for detailed analysis. This involved filtering the dataset to include only those entries where the trimming parameters indicated the use of a length-based filtering approach.

The filtered data was then sorted based on the specific length filter parameters, allowing for a coherent sequence when plotting. This sorting ensures that the visual representation accurately reflects the trend as the length filter parameter is adjusted.

Creating Figures for Comparative Analysis

A figure comprising two subplots was created to facilitate a comparative analysis of two key metrics: N50 and the number of N's per 100 kbp.

- The first subplot focuses on the variation of the N50 metric as the length filter parameter changes. It employs a line graph to connect data points, providing a clear visual representation of how N50 values adjust in response to changes in the length filter setting.

- The second subplot shifts attention to the change in the number of N's per 100 kbp, again utilizing a line graph to depict the relationship between the length filter parameter and this specific metric.

Both subplots include markers at data points, gridlines for easier reference, and legends for metric identification, enhancing the clarity and interpretability of the presented data.

A.2.8 Visualization with Matplotlib and Seaborn (Scatter Plot)

The visualizations (the code snippet A.8) were generated using Matplotlib, with specific attention to plot aesthetics such as color coding—blue for N50 and red for the number of N's per 100 kbp—to differentiate between the assembly metrics. The choice of line and marker styles ensures a seamless narrative of data trends across the length filter parameters.

```
# Code snippet for plotting
fig, axs = plt.subplots(1, 2, figsize=(20, 6))
# N50 Plot
axs[0].plot(length_filter_params, length_filter_data.loc[length_filter_sorted, 'N50'], marker='o', linestyle='-', color='blue', label='N50')
# Number of N's per 100 kbp Plot
axs[1].plot(length_filter_params, length_filter_data.loc[length_filter_sorted, "# N's per 100 kbp"], marker='o', linestyle='-', color='red', label="# N's per 100 kbp")
```

code snippet A.8: Scatter Plot Visualization of Metrics Variation

This thorough approach highlights the critical role of systematic data preparation and the power of visual analytics in revealing the effects of different genomic data trimming methods.

A.2.9 Data Filtering: The "Sliding Window" Trimming Method

The dataset was filtered to focus exclusively on the entries about the **"Sliding Window"** trimming method. This selective process is critical for analyzing the specific impact of window size and quality thresholds on the N Ratio.

The parameters associated with the **"Sliding Window"** method, namely window sizes and quality thresholds, were extracted and converted into numerical format. This preparation stage is essential for accurate and meaningful plotting.

A.2.10 Visualization with Plotly Using 3D Scatter Plot

A 3D scatter plot was created using Plotly to visualize (the code snippet A.9) the relationship between window size, quality threshold, and the N Ratio metric. This visualization technique provides a comprehensive view of the data, allowing for an in-depth analysis of how varying the window size and quality threshold parameters influence the N Ratio.

```
import plotly.graph_objects as go

# Create a Plotly 3D scatter plot for N50
fig = go.Figure(data=[go.Scatter3d(
    x=window_sizes,
    y=quality_thresholds,
    z=sliding_window_data['N_Ratio_x10k'],
    mode='markers',
    marker=dict(size=5, color='blue', opacity=0.8)
)])

fig.update_layout(title='3D Scatter Plot of N_Ratio_x10k for Sliding Window Trimming',
                  scene=dict(xaxis_title='Window Size',
                             yaxis_title='Quality Threshold',
                             zaxis_title='N_Ratio_x10k'))
fig.show(renderer="browser")
```

code snippet A.9: Creating 3D Scatter Plot with Plotly

This approach highlights the utility of 3D scatter plots in revealing complex relationships within the data, facilitated by Plotly's interactive visualization capabilities. The scatter plot effectively illustrates the dependency of the N Ratio on both the window size and quality threshold, providing insights that are crucial for optimizing the trimming process.

A video demonstration of this trimming method evaluation using 3D scatter plots is available for downloading at `https://github.com/shliamin/NGS-pipeline/blob/main/video/Sliding%20Window%20Data%20Visualisation.mov`.

A.3 Supplementary Literature Review

Throughout the development of the research, an extensive range of supplementary literature was reviewed to enhance the foundational understanding and methodological approach. These additional resources, not originally part of the primary literature collection, offered significant insights and technical advice pertinent to achieving the study's goals.

Delving into Nextflow, as detailed on its *official website* (`https://www.nextflow.io/index.html`), provided a comprehensive overview of this workflow tool's capabilities in facilitating scalable and reproducible scientific workflows. The integration of Nextflow is seen as essential for efficiently managing complex data analysis pipelines, highlighting its significant role in modern bioinformatics research.

Additionally, the study involved a thorough examination of the SPAdes algorithm, as described in the publication available on the *National Center for Biotechnology Information* (`https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3342519/`). This algorithm's role in facilitating accurate genome assemblies was critically assessed, highlighting its relevance to the research's genetic analysis component.

The dissertation by Alexey Gurevich, accessible via *Saint Petersburg State University's repository* (`https://disser.spbu.ru/files/2018/disser_gyrevich_aa.pdf`), and the documentation of his dissertation defense (`https://youtu.be/Hv8Bt7rVt8Q`) were also reviewed. This work on the QUAST tool provided valuable perspectives on quality assessment for genome Assemblies, enriching the research methodology with critical evaluation techniques for genomic data.

To broaden the literature base, the study was supplemented with a selection of books that provide comprehensive insights into bioinformatics and genomic data analysis.

1. *Bioinformatics Data Skills* by Vince Buffalo (O'Reilly Media, 2015) - A practical guide to the skills needed to efficiently handle and analyze bioinformatics data, from data cleaning to advanced analyses.

2. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids* by Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison (Cambridge University Press, 1998) - A foundational text on the application of probabilistic models to biological sequences, providing essential methodologies for analyzing DNA, RNA, and protein sequences.

These resources were instrumental in broadening the theoretical base and methodological framework of the research, leading to a more detailed and nuanced comprehension of the study's subject matter.

Declaration of Authenticity

I hereby declare that this bachelor's thesis titled "**Development of a Pipeline for the Benchmarking of Next Generation Sequencing Quality Control Tools**" is my own original work and has been written by me in its entirety. I have clearly referenced any sources used in the study. This thesis has not been submitted for any degree or examination at any other university or institution.

This thesis was prepared at **Berlin University of Applied Sciences (HTW Berlin)** under the supervision of **Prof. Piotr Wojciech Dabrowski**. I have adhered to the **HTW Berlin** guidelines for integrity in academic research and understand the potential consequences of any breach of this policy.

I also confirm that the printed version of my thesis is identical to the submitted electronic version.

Efim Shliamin

Date: 04.02.2024

Place: Berlin

Student ID: s0573270

Program: Applied Computer Science

Z