Pedro Filipe
Carneiro Venâncio

Aplicação de novos algoritmos bioinformáticos na análise de dados de Next Generation Sequencing (NGS)

Application of new bioinformatic algorithms in Next Generation Sequencing (NGS) data analysis.

Pedro Filipe
Carneiro Venâncio

Aplicação de novos algoritmos bioinformáticos na análise de dados de Next Generation Sequencing (NGS)

Application of new bioinformatic algorithms in Next Generation Sequencing (NGS) data analysis.

"*Sometimes it's the people no one can imagine anything of who do the things no one can imagine*"

— Alan Turing, mathematician and computer scientist

Universidade de Aveiro
2024

Pedro Filipe Carneiro Venâncio

Aplicação de novos algoritmos bioinformáticos na análise de dados de Next Generation Sequencing (NGS)

Application of new bioinformatic algorithms in Next Generation Sequencing (NGS) data analysis.

Relatório de estágio curricular apresentado à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Bioinformática Clínica, especialização em Bioinformática do Genoma , realizado sob a orientação científica da Doutora Gabriela Maria Ferreira Ribeiro de Moura, Professora auxiliar do Departamento de Ciências Médicas da Universidade de Aveiro, e supervisão da Doutora Alexandra Filipa Lopes, membro da entidade de acolhimento Unilabs.

Dedico este trabalho à Bárbara e à minha família e amigos.

o júri / the jury

presidente / president

Prof. Doutor João Antunes da Silva
professor associado da Faculdade de Engenharia da Universidade do Porto

vogais / examiners committee

Prof. Doutor João Antunes da Silva
professor associado da Faculdade de Engenharia da Universidade do Porto

Prof. Doutor João Antunes da Silva
professor associado da Faculdade de Engenharia da Universidade do Porto

Prof. Doutor João Antunes da Silva
professor associado da Faculdade de Engenharia da Universidade do Porto

Prof. Doutor João Antunes da Silva
professor associado da Faculdade de Engenharia da Universidade do Porto

Prof. Doutor João Antunes da Silva
professor associado da Faculdade de Engenharia da Universidade do Porto

agradecimentos / acknowledgements

Agradeço à Alexandra Lopes, a minha orientadora na Unilabs, pela confiança que sempre demostrou no meu trabalho. A tua disponibilidade constante e a forma prática como me foste guiando ao longo deste percuso foram essenciais para chegar até aqui.

À Professora Gabriela Moura, a minha orientadora na Universidade de Aveiro, agradeço por me ter colocado no caminho certo e por ter acreditado em mim e neste projeto. A sua orientação foi fundamental para manter o foco e a direção certa.

Ao Professor Bruno Gago, Diretor do Mestrado, agradeço por nunca ter desistido de mim, mesmo quando o caminho parecia impossível. Obrigado por ter sido um guia e uma referência.

Ao Professor Miguel Pais Vieira, agradeço pela forma como me preparou ao longo deste ano para este desafio. A sua exigência e rigor foram essenciais para que este projeto fosse um sucesso. Obrigado por ter acreditado em mim.

A todos os professores com quem me cruzei. Obrigado por terem partilhado o vosso conhecimento e experiência, por terem sido uma inspiração e por terem contribuído para a minha formação.

Ao Ricardo Pais, um obrigado especial pela persistência e pela paciência. Obrigado por ter estado sempre disponível para ajudar e por ter sido um verdadeiro salvavidas a resolver cada problema que foi surgindo (e foram muitos).

À Béryl Royer-Bertrand, por, mesmo à distância, ter dado um contributo essencial através do seu vasto conhecimento em bioinformática. A sua ajuda foi crucial em momentos chave, e sou grato por isso.

Ao Alberto Pessoa, meu colega e amigo, um grande obrigado pelo apoio, pelas discussões francas e pelas críticas construtivas que ajudaram a dar forma a este trabalho.

Aos meus colegas da Unilabs, obrigado por me terem acolhido tão bem e por tornarem esta experiência mais leve e enriquecedora. O ambiente de partilha e camaradagem foi essencial para que este percurso fosse tão positivo.

Aos meus amigos de Bragança e de Aveiro, agradeço por estarem sempre presentes, por me apoiarem quando foi preciso e por nunca deixarem de acreditar nesta amizade, mesmo quando estive mais ausente.

À família que me adotou através da Bárbara, o meu obrigado pelo carinho e apoio incondicional. Vocês tornaram este caminho muito mais fácil e agradável com a vossa amizade e alegria.

À minha família, ao meu pai, à minha mãe e às minhas avós, pela oportunidade que me deram e por nunca desistirem de mim, mesmo quando parecia que o melhor era desistir. A eles devo a educação e a pessoa em que me tornei. Obrigado pelo sacrifício. Valeu a pena!

E, por fim, à Bárbara. Obrigado por estares sempre ao meu lado, por seres o meu apoio nos dias bons e nos dias menos bons. Deste-me força quando precisei, foste a voz da razão quando perdi o norte e o ombro onde me apoiei em todas as fases deste percurso. Sem ti, este projeto teria sido muito mais difícil. Esta conquista também é tua.

Palavras Chave

Sequenciação de Nova Geração (NGS), Bioinformática, Análise de Dados Genómicos, Profundidade de Cobertura, Amplitude de Cobertura, Samtools, Painéis de Genes, Exoma, Gene Único

Resumo

Os avanços na Sequenciação de Nova Geração (NGS) transformaram a medicina genómica, permitindo uma análise de dados genéticos rápida e acessível. No entanto, a deteção precisa de variantes genéticas e a avaliação da qualidade da sequenciação continuam a ser desafios no diagnóstico clínico. Esta tese apresenta o desenvolvimento de um software que automatiza o cálculo de métricas chave de NGS, como a profundidade de cobertura e a percentagem de cobertura em diversos limiares. O software utiliza ferramentas bioinformáticas como SAMtools e integra uma interface gráfica desenvolvida em Streamlit, permitindo a análise de painéis de genes, exoma e genes individuais, com flexibilidade na configuração e exportação de resultados.

Testado em grandes conjuntos de dados genómicos, o software calculou com sucesso as métricas necessárias, proporcionando informações detalhadas sobre a qualidade dos dados. A ferramenta demonstrou eficácia na identificação de regiões de baixa cobertura, facilitando a deteção de variantes genéticas e melhorando a acessibilidade para clínicos. Ao garantir uma cobertura abrangente das regiões-alvo, o software aumenta a fiabilidade na análise de NGS, apoiando diagnósticos mais precisos de doenças genéticas e raras.

Link para o repositório do projeto: `https://github.com/pfcv99/metrics_app.git`

Keywords Next Generation Sequencing (NGS), Bioinformatics, Genomic Data Analysis, Depth of Coverage, Breadth of Coverage, Samtools, Gene Panels, Exome, Single Gene

Abstract

Advancements in Next Generation Sequencing (NGS) have transformed genomic medicine, enabling rapid and accessible analysis of genetic data. However, the accurate detection of genetic variants and the assessment of sequencing quality remain challenges in clinical diagnostics. This thesis presents the development of software that automates the calculation of key NGS metrics, such as depth of coverage and percentage of coverage at various thresholds. The software utilizes bioinformatics tools like SAMtools and integrates a graphical interface developed in Streamlit, allowing for the analysis of gene panels, exomes, and individual genes, with flexibility in configuration and result exportation.

Tested on large genomic datasets, the software successfully calculated the required metrics, providing detailed insights into data quality. The tool demonstrated effectiveness in identifying low coverage regions, facilitating the detection of genetic variants and improving accessibility for clinicians. By ensuring comprehensive coverage of target regions, the software enhances reliability in NGS analysis, supporting more accurate diagnoses of genetic and rare diseases.

Link to the project repository: `https://github.com/pfcv99/metrics_app.git`

Acknowledgement of use of AI tools

Recognition of the use of generative Artificial Intelligence technologies and tools, software and other support tools.

I acknowledge the use of ChatGPT 3.5 (Open AI, https://chat.openai.com) for paraphrasing and translations, the use of Adobe Illustrator (Adobe, https://www.adobe.com/pt/products/illustrator) for image creation and editing, the use of FreePik (FreePik, https://br.freepik.com/) for vector images download, and the use of diagrams.net (diagrams.net, https://app.diagrams.net/) for diagram creation.

Contents

Contents i

List of Figures v

List of Tables vii

List of Code Snippets x

Glossary xi

1 Introduction 1

1.1 Internship Context and Framework . . . 1

1.2 Project motivation and objectives . . . 1

1.3 Document Structure . . . 2

1.4 Characterization of the Host Entity and Work Plan . . . 3

1.4.1 Unilabs Portugal . . . 3

1.4.2 Unilabs Genetics . . . 4

1.4.3 Timeline . . . 4

1.5 Theoretical Framework - Genetics and Genomics . . . 5

1.5.1 Evolution of genetics over the years . . . 5

1.5.2 Genetics vs Genomics . . . 6

1.5.3 Structure and function of DNA, RNA, and proteins . . . 6

1.5.4 Molecular Structure of DNA . . . 7

1.5.5 Discovery of the Double Helix . . . 7

1.5.6 DNA Packaging in Eukaryotic Cells . . . 8

1.5.7 Mutations and genetic variations . . . 8

1.6 Theoretical Framework - Sequencing Methods and Characteristics . . . 9

1.6.1 Next Generation Sequencing - Gene Panels, Exome, and Genome Sequencing 9

1.6.2 Next Generation Sequencing - Overview . . . 10

1.7 Theoretical Framework - Bioinformatics . . . 11

1.7.1 Next Generation Sequencing - Data Analysis . . . 11
1.7.2 Next Generation Sequencing - Validation . . . 17

2 Software development process **19**
2.1 Requirements . . . 19
2.1.1 Functional Requirements . . . 19
2.1.2 Non-Functional Requirements . . . 20
2.2 System Design and Architecture . . . 21
2.2.1 User Workflow . . . 21
2.3 Development . . . 25
2.3.1 Environment preparation . . . 25
2.3.2 Streamlit . . . 26
2.3.3 Universal Browser Extensible Data (BED) - Processing and Simplification of BED Files for Genomic Analysis and Integration of Matched Annotation from NCBI and EMBL-EBI (MANE) Transcripts . . . 27
2.3.4 Samtools Depth Calculation in Python with Streamlit . . . 31
2.3.5 Detailed Breakdown of the Python Script for Metrics Calculation . . . 35
2.3.6 Results Generation and Display Functionality . . . 44
2.4 Deployment . . . 54
2.4.1 Base Image and Working Directory . . . 55
2.4.2 Installing System Dependencies . . . 55
2.4.3 Installing Miniconda and Python Environment Setup . . . 56
2.4.4 Installing Samtools . . . 56
2.4.5 Exposing the Application Port and Healthcheck . . . 57
2.4.6 Entry Point and Application Execution . . . 57

3 Results **59**
3.1 Evolution of Software Development . . . 59
3.1.1 Initial Stages: Basic Functionality and User Interaction . . . 59
3.1.2 Refinement: Introducing Flexibility and Multiple Analysis Modes . . . 60
3.1.3 Overview of the Final Version . . . 61
3.2 Test and Validation . . . 68
3.2.1 Single Gene Analysis . . . 69
3.2.2 Gene Panel Analysis . . . 69
3.3 Performance . . . 70
3.4 Users feedback . . . 70

4 Additional activities during the internship **73**
4.1 Additional Activities during the Internship . . . 73

5 Discussion **75**

5.1 SWOT Analysis . . . 75

5.1.1 Strengths . . . 75

5.1.2 Weaknesses . . . 76

5.1.3 Opportunities . . . 77

5.1.4 Threats . . . 77

5.2 Software otimizations . . . 78

5.2.1 Parallel and Distributed Processing with Dask . . . 78

5.2.2 Integration with AWS S3 Storage Service . . . 78

5.2.3 Improvements to the Graphical User Interface . . . 78

5.2.4 Secure and Efficient Authentication and Authorization System . . . 78

5.2.5 Enhanced Software Documentation . . . 78

5.2.6 Implementation of Automated Testing . . . 79

5.2.7 Code Optimization for Efficiency and Execution Time . . . 79

5.2.8 Implementation of Monitoring and Alert Systems . . . 79

5.2.9 Impact of Matched Annotation from NCBI and EMBL-EBI Transcripts on Software Metrics . . . 79

5.3 Impact on the company . . . 80

6 Final remarks **83**

Bibliography **85**

A Additional content **87**

List of Figures

1.1 Intership Timeline: A visual representation of the key milestones and deadlines met during the internship. 5
1.2 Evolution of genetics over the years: A brief timeline with some of the major historical milestones. Image adapted from [7] and [3] 6
1.3 Representation of Deoxyribonucleic Acid (DNA) and its constituents. In the top left corner, the nitrogenous bases are illustrated, categorized into Purines (Guanine and Adenine) and Pyrimidines (Thymine and Cytosine). Below this, on the left side, the paired nitrogenous bases are shown, along with their respective hydrogen bonds and the sugar-phosphate backbone connections. On the right side of the image, a chromosome is depicted unraveling, revealing the DNA structure. [11] 7
1.4 A figure with the reconstruction of the Watson-Crick's double helix model of DNA in 1.4a built by Science Museum Group Collection [12] and Franklin's X-ray diagram of the B form of sodium thymonucleate (DNA) fibres in 1.4b, published in Nature on 25 April 1953 [13] 8
1.5 Visualization of cluster intensities in 2-Channel Sequencing (NovaSeq 6000 - Illumina). The image shows the intensity data for each cluster from both the red and green channels, with each cluster representing a different DNA base. Image from [20]. 12
1.6 FASTQ file format example. The image shows a read identifier, nucleotide sequence, and quality score string in a FASTQ file. The P error calculation was performed based on the Phred quality score equation and Quality Score Encoding from Table A.2. This file example was adapted from [22]. 13

1.7 Overview of the bioinformatics pipeline for next-generation sequencing (Next Generation Sequencing (NGS)) analysis. DNA fragments with adapter sequences are processed to generate raw sequence reads (FASTQ). After quality control and adapter trimming, analysis-ready FASTQ files are aligned to a reference genome, producing aligned Binary Alignment Map (BAM)Compressed Reference-oriented Alignment Map (CRAM) files. These are further processed to identify sequence variants, including Single-Nucleotide Variants (SNVs), Insertions and Deletions (INDELs), and structural variants such as translocations, duplications, and inversions. The identified variants are compiled in a Variant Call Format (VCF) file, which undergoes annotation and interpretation to generate a clinical report. Adapted from [23], [24], [25], [26] 16
1.8 Scheme related to Coverage/Breadth of Coverage and Read Depth/Depth of Coverage in a gene. In this case, approximately 10% of the gene is depicted as not covered, and the depth reaches up to 9x. Adapted from [39] 18

2.1 Scheme of the software architecture. 23
2.2 Software directory structure. 24

3.1 First version of the Graphical User Interface (GUI). 60
3.2 Second version of the GUI. 61
3.3 Login Interface for the Software 62
3.4 Single Gene Analysis Workflow 63
3.5 Results Tab Loading the Final Report 63
3.6 Final Report with Detailed Metrics for Gene TP53 64
3.7 Depth of Coverage Visualization for Gene TP53 65
3.8 Gene Panel Input Selection and Submission 66
3.9 Overall Gene Panel Results for Hereditary Breast and Ovarian Cancer 67
3.10 Detailed Metrics for the BRCA1 Gene 67
3.11 Exon-Level Metrics for the BRCA1 Gene 68

4.1 Gene Panel Creator Interface 74

List of Tables

1.1 Base Calls in 2-Channel Sequencing (NovaSeq 6000 - Illumina). Table from [20] 13

3.1 Comparison of Metrics between the developed software and Omnomics for Gene TP53 . . 69
3.2 Comparison of Metrics between the developed software and Omnomics for Gene Panel: Cancro da mama e ovário (27 genes) 70

A.1 Unilabs test catalog 87
A.2 Quality score encoding. Adapter from [55] 88
A.3 Packages used in the project 89

List of Code Snippets

1 Python script for converting BED files to JavaScript Object Notation (JSON) format. 29
2 Loading mappings for gene identifiers and MANE equivalence. 29
3 Processing JSON files to generate a simplified BED file. 30
4 Assigning exon numbers and writing to output BED file. 31
5 Python function for calculating Depth of Coverage using Samtools. 32
6 Session state initialization. 32
7 Validation and setup for gene and exon selection. 33
8 Filtering the BED file based on gene and exon selections. 34
9 Running Samtools and handling depth output. 35
10 Example usage of the depth function. 35
11 Importing necessary libraries. 36
12 Defining the metrics initialization function. 37
13 Accessing filtered BED and depth data from session state. 37
14 Reading filtered BED content into a DataFrame. 37
15 Processing depth data for each BAM/CRAM file. 38
16 Calculating metrics for all genes combined. 39
17 Calculating depth of coverage percentages for different thresholds. 39
18 Calculating counts for specific depth intervals. 40
19 Calculating metrics for each gene. 41
20 Calculating depth percentages and intervals for each gene. 42
21 Calculating metrics for each exon within each gene. 43
22 Storing and returning the calculated metrics. 44
23 Importing necessary libraries and modules for the script. 44
24 Defining logo file paths and displaying them in the application. 45
25 Defining the ‘render_metric_filters‘ function for metric selection. 45
26 Defining the ‘select_all_metrics‘ function to toggle metric selection. 46
27 Defining the desired metrics order for display. 46
28 Calculating metrics and obtaining results for samples. 46
29 Preparing the DataFrame for all genes metrics. 47

30 Preparing individual DataFrames for each gene. 47
31 Preparing DataFrames for exon-level metrics. 48
32 Generating and downloading a Portable Document Format (PDF) report for the selected sample. 50
33 Creating tabs based on the type of analysis. 50
34 Displaying metrics in the "Overview" tab with filters. 52
35 Displaying metrics in the "Gene Detail" tab with filters. 53
36 Displaying metrics in the "Exon Detail" tab with filters. 54
37 Dockerfile: Setting the base image and working directory. 55
38 Dockerfile: Installing system dependencies. 56
39 Dockerfile: Installing Miniconda and setting up the Python environment. 56
40 Dockerfile: Installing Samtools. 57
41 Dockerfile: Exposing application port and setting up health checks. 57
42 Dockerfile: Setting entry point and application execution. 57

Glossary

NGS	Next Generation Sequencing
CLIA	Clinical Laboratory Improvement Amendments
ISO	International Organization for Standardization
WES	Whole Exome Sequencing
WES	Whole Genome Sequencing
aCGH	Comparative Genomic Hybridization
FISH	Fluorescence In Situ Hybridization
MLPA	Multiplex Ligation-Dependent Probe Amplification
DNA	Deoxyribonucleic Acid
HGP	Human Genome Project
RNA	Ribonucleic Acid
SWOT	Strengths Weeknesses Opportunities and Threats
WSL	Windows Subsystem for Linux
ES	Exome Sequencing
GS	Genome Sequencing
GRCh38/hg38	Reference Consortium Human Build 38
BAM	Binary Alignment Map
SAM	Sequence Alignment Map
CRAM	Compressed Reference-oriented Alignment Map
VUS	Variants With Unknown Significance
NLP	Natural Language Processing
SNPs	Single-Nucleotide Polymorphisms
SNVs	Single-Nucleotide Variants
INDELs	Insertions and Deletions
CNVs	Copy Number Variants
VAF	Variant Allele Frequency
REF	Reference Allele
ALT	Alternate Allele
GQ	Genotype Quality
VCF	Variant Call Format
gVCF	Genomics Variant Call Format
MAF	Mutation Annotation Format
BED	Browser Extensible Data
CSV	Comma-Separated Values
PDF	Portable Document Format
GDPR	General Data Protection Regulation
FR	Functional Requirements
NFR	Non-Functional Requirements
MANE	Matched Annotation from NCBI and EMBL-EBI
GUI	Graphical User Interface
BP	Base Pair
JSON	JavaScript Object Notation

CHAPTER 1

Introduction

"The only source of knowledge is experience." - Albert Einstein, theoretical physicist

1.1 Internship Context and Framework

This document represents the final report of the internship carried out as part of the Internship Curricular Unit (49991) of the second year of studies of the Master's Degree in Clinical Bioinformatics, with specialization in Genome Bioinformatics, at the University of Aveiro. The internship lasted nine months, starting on November 21st, 2023, and ending on July 19th, 2024, totalling 1296 hours of work in the company Unilabs Portugal, specifically in the Unilabs Genetics department.

During this period, the trainee had the opportunity to apply the knowledge acquired throughout the course and to get involved in practical projects related to bioinformatics and genomics. Unilabs, a recognized company in the health area, provided a professional environment where the intern could collaborate with experienced professionals and actively participate in projects relevant to clinical bioinformatics. This report addresses the activities developed during the internship and the contributions to the projects in which the intern was involved.

This introductory section aims to offer an overview of the context in which the internship was carried out, laying the foundations for understanding the activities and results presented throughout the report.

1.2 Project motivation and objectives

Currently, Unilabs uses a genomic intelligence platform that uses Natural Language Processing (NLP) to analyse new scientific publications of a genetic nature and incorporate them into an always updated knowledge base. This platform is particularly useful in prioritizing sequenced variants, for genetic diagnosis purposes, in their interpretation and in the production of clinical reports, thus enabling the provision of increasingly personalized care.

However, at the time of this internship, Unilabs was in the migration phase to this new platform and, therefore, as a complementary strategy, a new independent software was developed to obtain the necessary metrics for genomic analyses, not directly provided by the aforementioned platform, ensuring compliance with the guidelines and practices recommended for NGS. These metrics are important for assessing data quality, i.e., they indicate how well the target regions were covered by sequencing. In the case of the present stage, it was suggested to obtain the sequencing Depth of Coverage at different thresholds. In addition, the Depth of Coverage directly influences the ability to detect genetic variants: regions with low Depth of Coverage can result in undetected or underestimated variants. Additionally, coverage metrics are also useful to optimize sequencing protocols, adjusting experimental parameters to ensure adequate coverage of target regions and minimize unnecessary costs.

As explained in detail below, the software created and described in this report allows the obtaining of Average Read Depth, Breadth of Coverage and Depth of Coverage at 1x, 10x, 15x, 20x, 30x, 50x, 100x, and 500x per gene and per panel in analysis of gene panels. Additionally, in addition to the presentation of metrics by panel, single gene and exome analysis was also implemented.

1.3 Document Structure

This document is organized into six main chapters, each comprising several sections and subsections, followed by a bibliography and additional content.

The first chapter, **Introduction**, lays the groundwork for the thesis by providing the **Internship Context and Framework**, which offers an overview of the internship setting and its significance. It then details the **Project Motivation and Objectives**, outlining the purpose and goals of the work. This is followed by the **Document Structure** section, which explains the organization of the document. The **Characterization of the Host Entity and Work Plan** section describes the host entities—**Unilabs Portugal** and **Unilabs Genetics**and presents the **Timeline** of the work plan. The chapter concludes with the **Theoretical Framework**, divided into three main parts: **Genetics and Genomics**, **Sequencing Methods and Characteristics**, and **Bioinformatics**. Each part delves into specific topics such as the **Evolution of Genetics**, **Genetics vs. Genomics**, **Structure and Function of DNA, Ribonucleic Acid (RNA), and Proteins**, the **Molecular Structure of DNA**, the **Discovery of the Double Helix**, **DNA Packaging in Eukaryotic Cells**, **Mutations and Genetic Variations**, **Next Generation Sequencing - Gene Panels, Exome, and Genome Sequencing**, **Next Generation Sequencing - Overview**, **Next Generation Sequencing Data Analysis** and **Next Generation Sequencing - Validation**.

The second chapter, **Software Development Process**, details the methodology and implementation of the software project. It begins with the **Requirements** section, specifying both **Functional** and **Non-Functional Requirements** of the software. The **System Design and Architecture** section discusses the overall design, including the **User Workflow**. The **Development** section elaborates on the implementation details, covering topics such as **Environment Preparation**, **Streamlit**, **Processing and Simplification of BED Files**

for Genomic Analysis, **SAMtools Depth Calculation in Python with Streamlit**, **Detailed Breakdown of the Python Script for Metrics Calculation**, and **Results Generation and Display Functionality**. The chapter concludes with the **Deployment** section, explaining how the software was deployed for use.

The third chapter, **Results**, presents the outcomes of the software development process. It discusses the **Evolution of Software Development**, highlighting the **Initial Stages: Basic Functionality and User Interaction**, the **Refinement: Introducing Flexibility and Multiple Analysis Modes**, and provides an **Overview of the Final Version**. The chapter also covers **Test and Validation** procedures, including **Single Gene Analysis** and **Gene Panel Analysis**. It examines the software's **Performance**, offers a **Comparison with Other Tools**, and includes **User Feedback**.

The fourth chapter, **Additional Activities During the Internship**, describes other tasks and experiences undertaken during the internship, namely the creation of complementary tools.

The fifth chapter, **Discussion**, provides a comprehensive analysis of the work. It includes a **SWOT Analysis** of the software, discussing its **Strengths**, **Weaknesses**, **Opportunities**, and **Threats**. The chapter also suggests **Software Optimizations**, covering topics like **Parallel and Distributed Processing with Dask**, **Integration with AWS S3 Storage Service**, **Improvements to the Graphical User Interface**, **Secure and Efficient Authentication and Authorization System**, **Enhanced Software Documentation**, **Implementation of Automated Testing**, **Code Optimization for Efficiency and Execution Time**, **Implementation of Monitoring and Alert Systems**, and the **Impact of MANE Transcripts on Software Metrics**. It concludes by discussing the **Impact on the Company**.

The sixth and final chapter, **Final Remarks**, summarizes the conclusions drawn from the work and offers suggestions for future improvements and new features for subsequent versions of the software.

The document concludes with a **Bibliography** listing all consulted and cited sources, followed by **Additional Content** in the appendix, which includes supplementary material relevant to the work.

1.4 Characterization of the Host Entity and Work Plan

1.4.1 Unilabs Portugal

Since the beginning of 2006, Unilabs has established solid roots in Portugal. It began its journey with the acquisition of most of the shares of the company "Medicina Laboratorial Dr. Carlos Torres" and since then it has grown steadily, following a strategy of acquiring high-quality laboratories and partners throughout the country. [1]

It has more than 3,500 employees and more than 500 doctors, and operates in more than 1,000 service units, performing more than 25 million medical procedures per year. [1]

In 2017, Unilabs took an important step by acquiring BASE Holding. With this acquisition, it has expanded its service offering to include radiology, affirming its position as a national leader in integrated clinical diagnostics and the provision of Complementary Diagnostic and Therapeutic Means. [1]

Currently, it offers a wide variety of services in various areas, including Clinical Analysis, Pathological Anatomy, Cardiology, Gastroenterology, Medical Genetics, Nuclear Medicine and Radiology. The company maintains its commitment to being close to people, providing answers that contribute to a healthier future. [1]

1.4.2 Unilabs Genetics

Unilabs Genetics (formerly called CGC Genetics) was founded three decades ago and was the first private Medical Genetics laboratory in Portugal. It has been present on the national scene regarding diagnosis through genetic studies. It has a wide selection of tests and is known for its collaborative and thorough approach, meeting the demands of clinicians in a variety of specialist areas. [2]

It is a leader in Europe in Medical Genetics, with special emphasis on rare diseases. It provides accurate diagnoses for public and private healthcare institutions, providing physicians and patients with detailed information about the nature of diseases, prognosis, and treatment options. In addition, the company supports academic institutions, research centres and the pharmaceutical industry with data and knowledge that contributes to the discovery of biomarkers and the development of new drugs. [2]

The Unilabs Genetics laboratory is located in the city of Porto and combines advanced technologies, bioinformatics and artificial intelligence, with a highly qualified team of medical geneticists, specialists in genetic counselling and laboratory technicians. The company follows the strictest quality and ethics policies, having certifications (Clinical Laboratory Improvement Amendments (CLIA), International Organization for Standardization (ISO) 15189, ISO 9001) that guarantee excellence in its services. [2]

The Table A.1 presents the test catalog provided by Unilabs Genetics.

1.4.3 Timeline

The internship at Unilabs followed a structured timeline, aimed at ensuring the successful completion of all assigned tasks. The timeline was designed to provide a clear framework for the development and implementation of various activities throughout the internship period. This detailed plan served as a guide to ensure that each phase of the project was completed efficiently and on time, while also allowing the necessary flexibility to accommodate any adjustments. The Figure 1.1 outlines the key milestones and deadlines that were met during this internship.

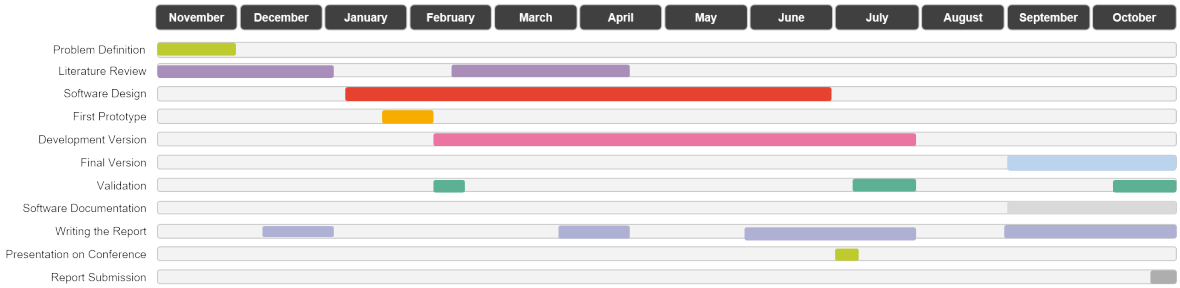


Figure 1.1: Intership Timeline: A visual representation of the key milestones and deadlines met during the internship.

1.5 Theoretical Framework - Genetics and Genomics

1.5.1 Evolution of genetics over the years

The history of genetics formally began in 1865 with Gregor Mendel's work on plant hybridization. However, the term "genetics" was only coined in 1906 by the English biologist William Bateson to define the new science of heredity. Based on Mendel's laws, genetics introduced groundbreaking concepts such as gene, genotype, and phenotype. By the 1910s, Mendelian genetics merged with the chromosomal theory of inheritance, giving rise to classical genetics. In this framework, the gene was seen as a unit of function, transmission, recombination, and mutation. [3]

This understanding persisted until the 1950s, when DNA was discovered as the material basis of heredity, marking the start of molecular biology. [4]

Following the discovery of DNA as hereditary material, molecular biology began to uncover the complexity of gene function. The fusion of Mendel's ideas with chromosomal theory also provided a more tangible understanding of genes, which could now be physically located on chromosomes. This integration led to significant advances, such as explaining Mendel's laws through cellular mechanisms and discovering genetic recombination. Genetics evolved into a more institutionalized science, with the establishment of academic chairs and specialized courses worldwide, solidifying its position as a central field in the biological sciences. [4]

The emergence of genomics in the latter half of the 20th century further transformed the field of genetics. The completion of the Human Genome Project (HGP) in 2003 [5], a milestone in genomics, revealed the entire sequence of human DNA, propelling the study of genes beyond individual units to entire genomes. This large-scale approach allowed scientists to explore the intricate network of genes and their interactions, significantly advancing our understanding of complex traits and diseases. Genomics also facilitated the development of personalized medicine, where treatments could be tailored based on an individual's genetic makeup. [4]

The discovery of the RNA-guided CRISPR-Cas9 system has made genome editing easier and more efficient. This breakthrough allows scientists to modify DNA in various cells and organisms with ease, removing previous experimental barriers. Today, CRISPR-Cas9 is widely used in basic research, biotechnology, and the development of new therapies. [6]

Figure 1.2 presents a timeline with some of the major historical milestones in the evolution of genetics over the years.

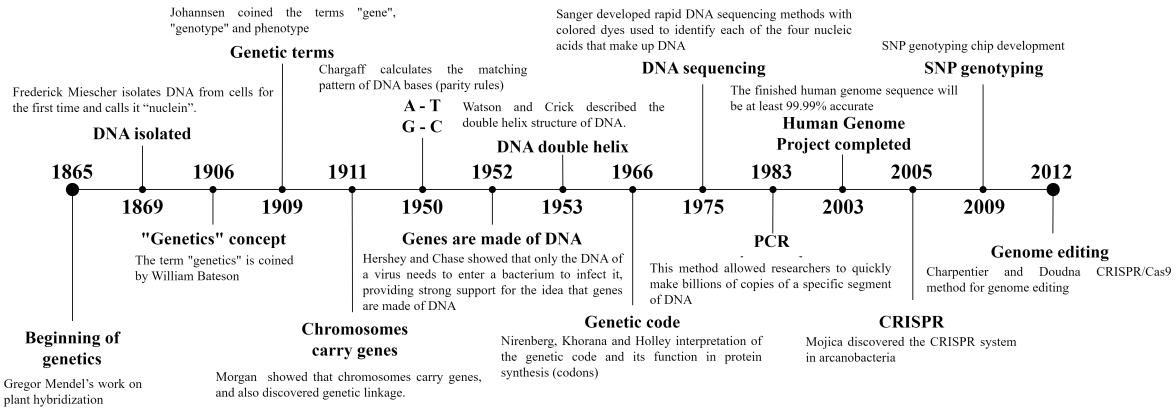


Figure 1.2: Evolution of genetics over the years: A brief timeline with some of the major historical milestones. Image adapted from [7] and [3]

1.5.2 Genetics vs Genomics

Genetics and genomics are both fields of study that explore the roles of genes in living organisms, but they focus on different aspects of heredity and DNA. Genetics is the study of specific genes and their influence on traits and conditions that are passed from one generation to the next. It examines how certain genes cause inherited disorders, such as cystic fibrosis and Huntington's disease. [8]

In contrast, genomics is a more recent field that encompasses the study of all the genes within an organism, referred to as the genome, and how these genes interact with each other and the environment. Unlike genetics, which focuses on individual genes, genomics uses advanced technologies like bioinformatics and high-performance computing to analyze vast amounts of genetic data. This comprehensive approach is crucial for studying complex diseases, such as cancer and diabetes, which result from the interplay between multiple genes and environmental factors. While both fields contribute to advancements in health and disease treatment, genomics represents a broader, more holistic view of genetic influence. [9]

1.5.3 Structure and function of DNA, RNA, and proteins

Nucleic acids, specifically DNA and RNA, are fundamental molecules in biological systems, playing key roles in storing and transmitting genetic information. DNA, which exists primarily as a double-stranded helix, encodes the instructions necessary for the growth, development, and reproduction of all living organisms. RNA, on the other hand, serves multiple purposes, including acting as a messenger that carries genetic information from DNA to the ribosomes for protein synthesis. The structural complexity and functional versatility of these molecules underscore their importance in the central dogma of molecular biology, which describes the flow of genetic information from DNA to RNA to proteins. [10]

1.5.4 Molecular Structure of DNA

The molecular structure of DNA is a polymer composed of repeating units called nucleotides. Each nucleotide consists of three components: a five-carbon sugar (deoxyribose), a phosphate group, and a nitrogenous base. The nitrogenous bases are categorized into two groups: purines (adenine and guanine) and pyrimidines (cytosine and thymine). The nucleotides are linked together by phosphodiester bonds, forming a sugar-phosphate backbone that gives DNA its structural integrity. DNA molecules are double-stranded, with two complementary strands running in opposite directions (antiparallel orientation). These strands are held together by hydrogen bonds between specific base pairs: adenine pairs with thymine, and guanine pairs with cytosine. This complementary base pairing is crucial for the accurate replication and transmission of genetic information. [10]

The Figure 1.3 shows a representation of the molecular structure of DNA and the base pairing between adenine (A), thymine (T), cytosine (C), and guanine (G).

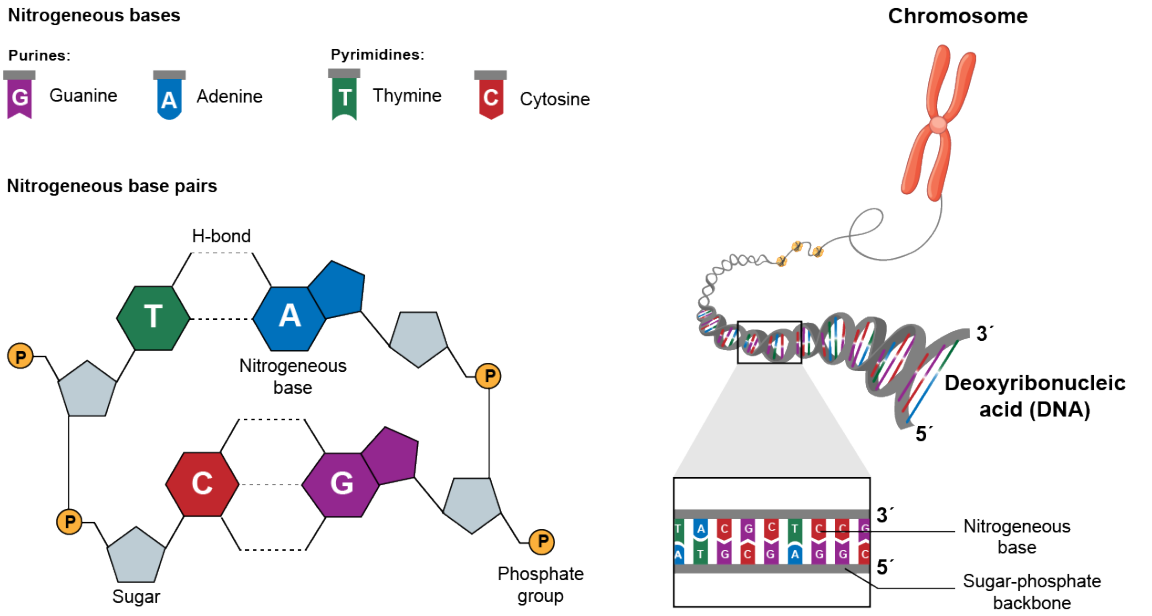
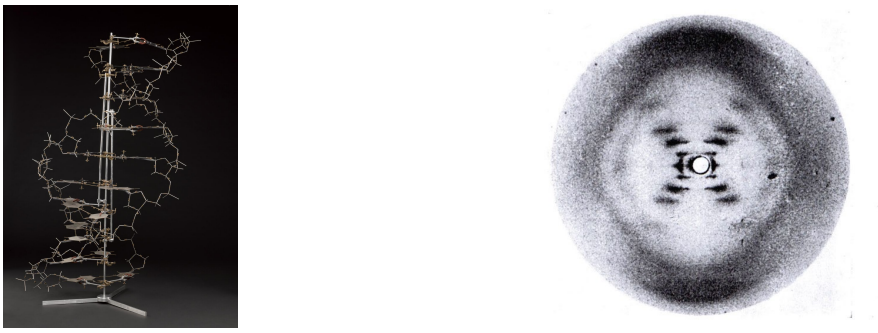


Figure 1.3: Representation of DNA and its constituents. In the top left corner, the nitrogenous bases are illustrated, categorized into Purines (Guanine and Adenine) and Pyrimidines (Thymine and Cytosine). Below this, on the left side, the paired nitrogenous bases are shown, along with their respective hydrogen bonds and the sugar-phosphate backbone connections. On the right side of the image, a chromosome is depicted unraveling, revealing the DNA structure. [11]

1.5.5 Discovery of the Double Helix

The three-dimensional structure of DNA, famously known as the double helix, was elucidated by James Watson and Francis Crick in 1953. Their discovery was informed by Rosalind Franklin's X-ray diffraction images, which revealed the helical structure of DNA. Watson and Crick proposed that the two strands of the helix are wound around each other, with the sugar-phosphate backbone on the outside and the nitrogenous bases on the inside.

The helical structure is right-handed, with ten base pairs per turn of the helix. The stability of the double helix is largely due to the hydrogen bonds between the complementary base pairs and the hydrophobic interactions between the stacked bases. This discovery not only explained how DNA could carry genetic information but also provided insights into how it could be replicated during cell division. [10]



(a) Double helix model of DNA **(b)** X-ray diagram of DNA

Figure 1.4: A figure with the reconstruction of the Watson-Crick's double helix model of DNA in 1.4a built by Science Museum Group Collection [12] and Franklin's X-ray diagram of the B form of sodium thymonucleate (DNA) fibres in 1.4b, published in Nature on 25 April 1953 [13]

1.5.6 DNA Packaging in Eukaryotic Cells

In eukaryotic cells, DNA is not free-floating within the nucleus; instead, it is highly organized and compacted into structures known as chromosomes. This compaction is achieved through the association of DNA with histone proteins, forming nucleosomes, which are the basic unit of chromatin. Each nucleosome consists of a segment of DNA wrapped around a core of histone proteins. These nucleosomes are further coiled and folded into higher-order structures, eventually forming the condensed chromosomes visible during cell division. The packaging of DNA into chromatin is essential for fitting the large eukaryotic genome into the limited space of the nucleus. Furthermore, chromatin structure plays a crucial role in gene regulation, as regions of tightly packed chromatin (heterochromatin) are generally transcriptionally inactive, while loosely packed regions (euchromatin) are more accessible to transcriptional machinery. [10]

1.5.7 Mutations and genetic variations

Mutations are fundamental drivers of genetic diversity, occurring when changes happen in the DNA sequence. These alterations can range from small-scale mutations, such as the substitution, insertion, or deletion of one or a few nucleotides, to large-scale mutations that involve significant chromosome segments or entire genes. Chromosome mutations specifically impact either individual nucleotides or larger chromosome fragments. They can involve deletions, insertions, inversions, translocations, or even gene duplications. On the other hand, genome mutations refer to changes in the number of whole chromosomes or sets of chromosomes and are studied separately. [14]

Genetic variation, refers to the observable differences between individuals within a population, which arise from variations in their genotypes. While mutations are the source of new genetic variation, genetic variation encompasses the broader concept of how different alleles at specific loci contribute to diversity within a population. This variation is shaped and refined by evolutionary forces such as natural selection, gene flow, and genetic drift. [14]

1.6 Theoretical Framework - Sequencing Methods and Characteristics

Even though there are several methods for sequencing DNA, the most widely used technique nowadays is Next Generation Sequencing. This section focuses only on NGS and its applications, including gene panels, exome sequencing, and genome sequencing.

1.6.1 Next Generation Sequencing - Gene Panels, Exome, and Genome Sequencing

Next Generation Sequencing technologies have revolutionized genomic medicine, enabling rapid advancements in clinical diagnostics, therapeutic decision-making, and disease prediction. Unlike traditional sequencing methods such as Sanger sequencing, which are limited by low throughput and high cost [15], NGS allows for the massively parallel sequencing of DNA, significantly increasing throughput and reducing costs by several orders of magnitude. As a result, clinical laboratories now have the capability to analyze nearly complete exomes or genomes of individuals, thereby improving the diagnostic process for a wide range of genetic conditions. [16]

NGS is characterized by the use of clonally amplified or single molecule templates, which are sequenced in parallel, providing an unprecedented ability to analyze large amounts of DNA efficiently. The adoption of NGS in clinical settings is growing rapidly, with three primary applications gaining prominence: disease-targeted gene panels, Exome Sequencing (ES), and Genome Sequencing (GS). [16]

Disease-targeted gene panels

Disease-targeted gene panels focus on a set of known disease-associated genes, allowing for greater Depth of Coverage and increased analytical sensitivity. This targeted approach improves the detection of heterozygous variants, mosaicism, or low-level heterogeneity, particularly in applications related to mitochondrial diseases or oncology. Targeted panels also allow laboratories to leverage desktop sequencers, reducing the costs of sequencing and data storage. However, follow-up techniques such as Sanger sequencing may be required to fill gaps in the data caused by regions of low coverage. [16]

Exome Sequencing

Exome Sequencing targets the coding regions of the genome, which constitute approximately 1-2% of the entire genome but harbor around 85% of known disease-causing mutations. [17] ES is particularly valuable in "detecting variants in known disease-associated genes as well as for the discovery of novel gene-disease associations" [16], with clinical studies demonstrating

a diagnostic success rate of approximately 20%. [18] Despite this potential, ES faces challenges related to coverage variability, as certain exonic regions may not be captured or sequenced with sufficient depth to make a sequence call, leading to the need for additional sequencing techniques to confirm findings. [16]

Genome Sequencing

Genome Sequencing offers a more comprehensive approach by covering both coding and non-coding regions of the genome. This technique simplifies the preparation of samples for sequencing by eliminating the need for pre-sequencing enrichment strategies. While GS holds promise for identifying regulatory variants and structural variants that are outside of coding regions, it remains the most expensive NGS technology and typically provides lower average depth of coverage. Nonetheless, as technology advances, these limitations are expected to diminish, making GS a more accessible option for clinical diagnostics. [16]

NGS involves three major components: sample preparation, sequencing, and data analysis. These steps are interrelated, and the quality of each influences the overall outcome of the sequencing process. Below is an overview of these essential stages according to the guidelines of [16].

1.6.2 Next Generation Sequencing - Overview

Sample preparation

The NGS process begins with the extraction of genomic DNA from a biological sample, typically from a patient. The quality and quantity of this DNA are critical to successful sequencing. Laboratories must specify the required sample type and amount based on their validation data. Sample mix-ups must be prevented through robust processes, as even minor errors can significantly affect results. [16]

For specific NGS applications like targeted panels or ES, enrichment strategies are employed to focus on a subset of genomic regions. Enrichment ensures that only the regions of interest are sequenced, improving efficiency and reducing costs. Enrichment can be achieved through various methods, including multiplex PCR and hybridization-based capture. [16]

Library Generation and Barcoding

Library generation is the process of preparing DNA fragments of a specific size (100-500 base pairs) for sequencing. These fragments are tagged with adapter sequences on both ends, which are essential for downstream sequencing steps. The fragmentation of DNA can be performed using different methods, each with its advantages and limitations. In most NGS workflows, PCR amplification is used to amplify the DNA library before sequencing. [16]

Barcoding is a crucial step in library preparation, where each sample is tagged with a unique sequence identifier. This allows multiple samples to be pooled together in a single sequencing run, reducing the cost per sample. Barcoding is typically integrated into the adapter sequences or added during a PCR enrichment step. [16]

Target Enrichment

In many NGS applications, particularly targeted panels and ES, only specific regions of the genome are sequenced. Target enrichment methods are used to isolate these regions before sequencing. Target enrichment strategies include PCR-based methods (e.g., single or multiplex PCR) and hybridization-based capture. While PCR-based methods are suitable for smaller panels, hybridization-based approaches are preferred for larger-scale applications like exome sequencing. [16]

Sequencing Platforms

NGS platforms are designed to perform millions of parallel chemical reactions, allowing them to sequence vast amounts of DNA simultaneously. These platforms utilize different sequencing chemistries, such as sequencing by synthesis, sequencing by ligation, and ion sensing. [19] The choice of sequencing platform depends on various factors, including sequence capacity, read length, run time, cost, and accuracy. [16]

Each platform has its own strengths and weaknesses. For example, some platforms excel at producing longer reads, while others are optimized for high-throughput sequencing at a lower cost per sample. The selection of a platform is influenced by the specific clinical or research application. [16]

1.7 Theoretical Framework - Bioinformatics

1.7.1 Next Generation Sequencing - Data Analysis

NGS generates an enormous amount of sequence data, necessitating the development of sophisticated data analysis pipelines. These pipelines are designed to process and interpret the raw sequencing data, transforming it into meaningful genomic information. NGS data analysis can be divided into four primary steps: Base Calling, Read Alignment, Variant Calling, and Variant Annotation. The bioinformatics analysis stage of NGS is essential for converting unprocessed sequencing data into biologically and clinically significant findings. This section provides a bioinformatics overview of these steps and their importance in the NGS workflow. [16]

Base Calling

Finding the nucleotide at each place in a sequencing read is known as **Base Calling**. In order to ensure that the raw data gathered during sequencing is transformed into a sequence of nucleotides, this step is usually included into the software of the sequencing device. [16]

For example, the Illumina NovaSeq 6000 Sequencing System uses two-channel sequencing approach, requiring only two images to represent data for all four DNA bases, with one image capturing information from the red channel and the other from the green channel. [20]

An 'N' designation, or 'no call' is used when a cluster fails to meet quality filters, registration is unsuccessful, or the cluster is not properly captured in the image. The process involves extracting intensity data for each cluster from both the red and green images and

comparing these intensities to identify four distinct populations. Each of these populations is associated with one of the DNA bases, and the base calling process assigns each cluster to the corresponding population. [20]

Figure 1.5 illustrates the intensity data for each cluster from both the red and green channels in 2-channel sequencing, as used in the NovaSeq 6000 Sequencing System.

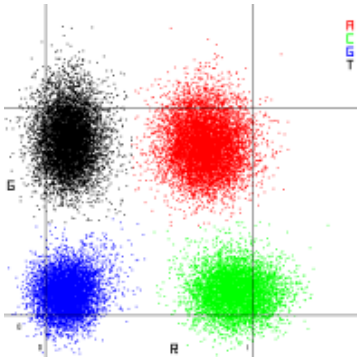


Figure 1.5: Visualization of cluster intensities in 2-Channel Sequencing (NovaSeq 6000 - Illumina). The image shows the intensity data for each cluster from both the red and green channels, with each cluster representing a different DNA base. Image from [20].

Table 1.1 outlines how the signal intensities from red and green fluorescence channels are used to identify DNA bases (A, C, G, T) during sequencing. Each base is associated with a unique combination of signal intensities from these two channels. The red and green channels either show intensity, marked as "on" (1), or no intensity, marked as "off" (0), at specific cluster locations where DNA bases are being read.

When both the red and green channels show intensity (1,1), this indicates the presence of adenine (A). The simultaneous detection of both red and green signals suggests that the cluster is emitting light in both spectrums, and this combination is uniquely attributed to adenine.

For cytosine (C), the red channel is on (1), but the green channel is off (0). This means that the cluster emits light in the red spectrum only, and the absence of green emission differentiates it as cytosine.

When neither the red nor the green channel shows intensity (0,0), the system identifies guanine (G). The lack of any signal at a known cluster location suggests that no fluorescence is being emitted, and this corresponds to guanine.

Finally, thymine (T) is identified when the red channel is off (0) and the green channel is on (1). In this case, the cluster is emitting light in the green spectrum only, and the absence of red fluorescence distinguishes it as thymine.

This method of combining red and green fluorescence signals allows the sequencing system to effectively differentiate between the four DNA bases by associating specific patterns of signal intensity with each base.

Table 1.1: Base Calls in 2-Channel Sequencing (NovaSeq 6000 - Illumina). Table from [20]

Base	Red Channel	Green Channel	Result
A	1 (on)	1 (on)	Clusters that show intensity in both the red and green channels.
C	1 (on)	0 (off)	Clusters that show intensity in the red channel only.
G	0 (off)	0 (off)	Clusters that show no intensity at a known cluster location.
T	0 (off)	1 (on)	Clusters that show intensity in the green channel only.

Read Alignment/Mapping

Base calling is followed by demultiplexing, the process of separating reads from different samples based on their unique barcodes. After demultiplexing, the next step is store reads in a FASTQ file, a text-based format that includes a read identifier, the nucleotide sequence, a separator, and a quality score string. The quality scores are calculated using the Phred scale, which represents the likelihood of a base call being correct, with higher scores indicating greater confidence. For a given base error probability, P, the Phred quality score, Q, is calculated in Equation 1.1.

$$Q = -10 \log_{10} P \tag{1.1}$$

FASTQ is the standard format for raw sequencing data and is commonly used in single-end and paired-end sequencing. [21] The Figure 1.6 shows an example of a FASTQ file, with the read identifier, nucleotide sequence, and quality score (highlighting the score for one T base and the respective P error).

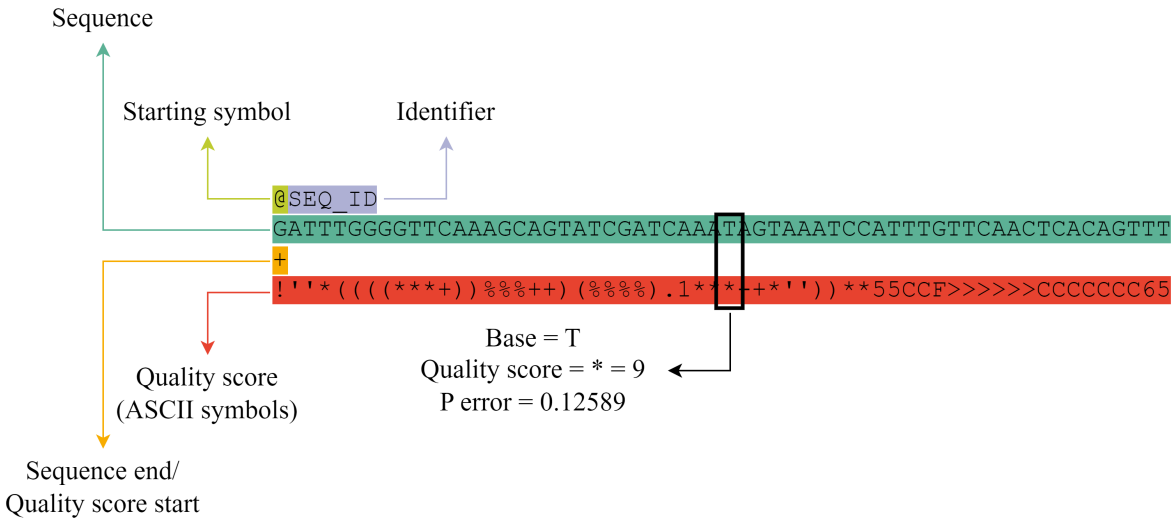


Figure 1.6: FASTQ file format example. The image shows a read identifier, nucleotide sequence, and quality score string in a FASTQ file. The P error calculation was performed based on the Phred quality score equation and Quality Score Encoding from Table A.2. This file example was adapted from [22].

Once the raw sequencing data was stored in a FASTQ file, the next step is **Read Alignment/Mapping**. [21] In this stage, the small DNA sequences (50–400 base pairs), reads, are positioned in relation to a reference genome, such as Reference Consortium Human

Build 38 (GRCh38/hg38). [16] To improve alignment accuracy, low-quality bases and adapter sequences are also trimmed and the results are saved in either Sequence Alignment Map (SAM) or BAM formats, with BAM being more efficient due to compression and indexing capabilities. A more compressed format, CRAM, further reduces file sizes by storing only differences from the reference genome, though it uses lossy compression in some cases (meaning that some CRAM files may not be fully convertible back to BAM) CRAM is becoming a popular choice for long-term data storage due to its space efficiency - a CRAM file could be 50%-80% smaller than a BAM file. [21]

Variant Calling

Variant Calling identifies genetic differences between a sample's sequencing reads and a reference genome using specialized algorithms known as variant callers. Common variants, include Single-Nucleotide Polymorphisms (SNPs), SNVs, small INDELs, and Copy Number Variants (CNVs) or alterations. The first tree are related to Sequence Variants, while the last one is related to Structural Variants. [23], [24], [25], [26] SNPs refer to single-base changes in germline DNA, often biallelic, while SNVs cover any point mutation. CNVs involve larger DNA segments that are amplified or deleted, and CNAs specifically refer to somatic changes, often observed in cancer. [21] There are also other types of variants, such as translocations, inversions, and complex rearrangements, which are less common but can have significant clinical implications. [23]

The confidence with which variants are called depends heavily on sequencing Depth of Coverage and the Variant Allele Frequency (VAF), which measures the proportion of DNA molecules in a sample that contain the variant allele. For germline heterozygous variants, where the variant is expected to appear in roughly 50% of reads, reliable detection can typically be achieved at sequencing depths of 20X to 30X. However, somatic variants, especially in cancer samples, tend to have lower VAFs due to tumor heterogeneity and sample contamination, requiring much higher coverage for reliable detection. [21]

Preprocessing of BAM files is essential before variant calling. Duplicates, originating from the same DNA fragment, must be marked to avoid skewing VAF estimates. Deduplication is recommended for whole-genome or exome sequencing but is usually skipped in PCR-based methods. Base quality scores are also recalibrated to correct systematic errors. After preprocessing, the BAM files are ready for variant detection. [21]

Germline Variant Calling for SNVs can be done using tools like Samtools or more advanced programs like GATK HaplotypeCaller, which improve accuracy in complex regions by applying local realignments. For biallelic SNPs, the Reference Allele (REF) is compared to the Alternate Allele (ALT). Humans, being diploid, have three possible SNP genotypes: homozygous reference (REF, REF), heterozygous (REF, ALT), and homozygous alternate (ALT, ALT), corresponding to VAFs of 0%, 50%, and 100%. Genotype Quality (GQ), similar to base quality scores, reflects the confidence in a genotype call, measured on a Phred scale. [21]

When detecting CNVs using NGS data, algorithms primarily rely on sequencing coverage

patterns, although they may also take into account the VAFs of overlapping variants. CNVs detection in targeted sequencing approaches like Whole Exome Sequencing (WES) or gene panels can be challenging due to coverage gaps and inconsistencies caused by capture bias. As a result, CNVs identification within individual samples is difficult, and many tools designed for this purpose compare the data against a reference set of normal samples. [21]

Somatic Variant Detection is enhanced by comparing tumor and matched normal tissue sequencing data, allowing inherited germline variants to be filtered out. When matched normal samples are unavailable, a "panel of normals" can serve as a reference. Additionally, classifiers trained on databases of somatic and germline variants can help predict the somatic status of variants found in tumor tissue. Despite these approaches, somatic variant detection still faces challenges, particularly due to the Euro-centric bias of many population allele frequency databases, which may reduce the accuracy of variant calls in underrepresented populations. [21]

Variants are typically stored in VCF files, which can include data from multiple samples. Genomics Variant Call Format (gVCF) files capture both variant and non-variant regions, allowing for easier data merging. Both VCF and gVCF files can be indexed for efficient access. Somatic mutations may also be saved in Mutation Annotation Format (MAF) files, which aggregate variant data from multiple VCFs. [21]

Variant Annotation

Once variants are called, **Variant Annotation** is carried out. This involves adding contextual information to each detected variant, such as determining its location within or near a gene and predicting its potential impact on protein function. Clinical interpretation of variants often includes data from variant databases, evolutionary conservation studies, and in silico predictions of pathogenicity. [16]

NGS often generates numerous variants, including many Variants With Unknown Significance (VUS), making it difficult to determine which are clinically relevant. Variant annotation helps prioritize and interpret these findings. For protein-coding regions, tools like REVEL [27] predict the functional impact of missense variants, while noncoding variants are assessed using resources such as CADD [28], FunSeq2 [29], and RegulomeDB [30]. These are complemented by data from projects like ENCODE [31] and Roadmap Epigenomics [32], and external databases like gnomAD [33], ClinVar [34], HGMD [35], and COSMIC [36]. Annotation software like ANNOVAR [37] integrates these resources to enrich VCF files with variant details. [21]

Overall, accurate and efficient data analysis in NGS requires significant bioinformatics support and robust computational infrastructure. This aspect of NGS is crucial for translating raw sequencing data into clinically relevant insights.

Figure 1.7 illustrates the general bioinformatics analysis pipeline for NGS, highlighting the key steps mentioned previously.

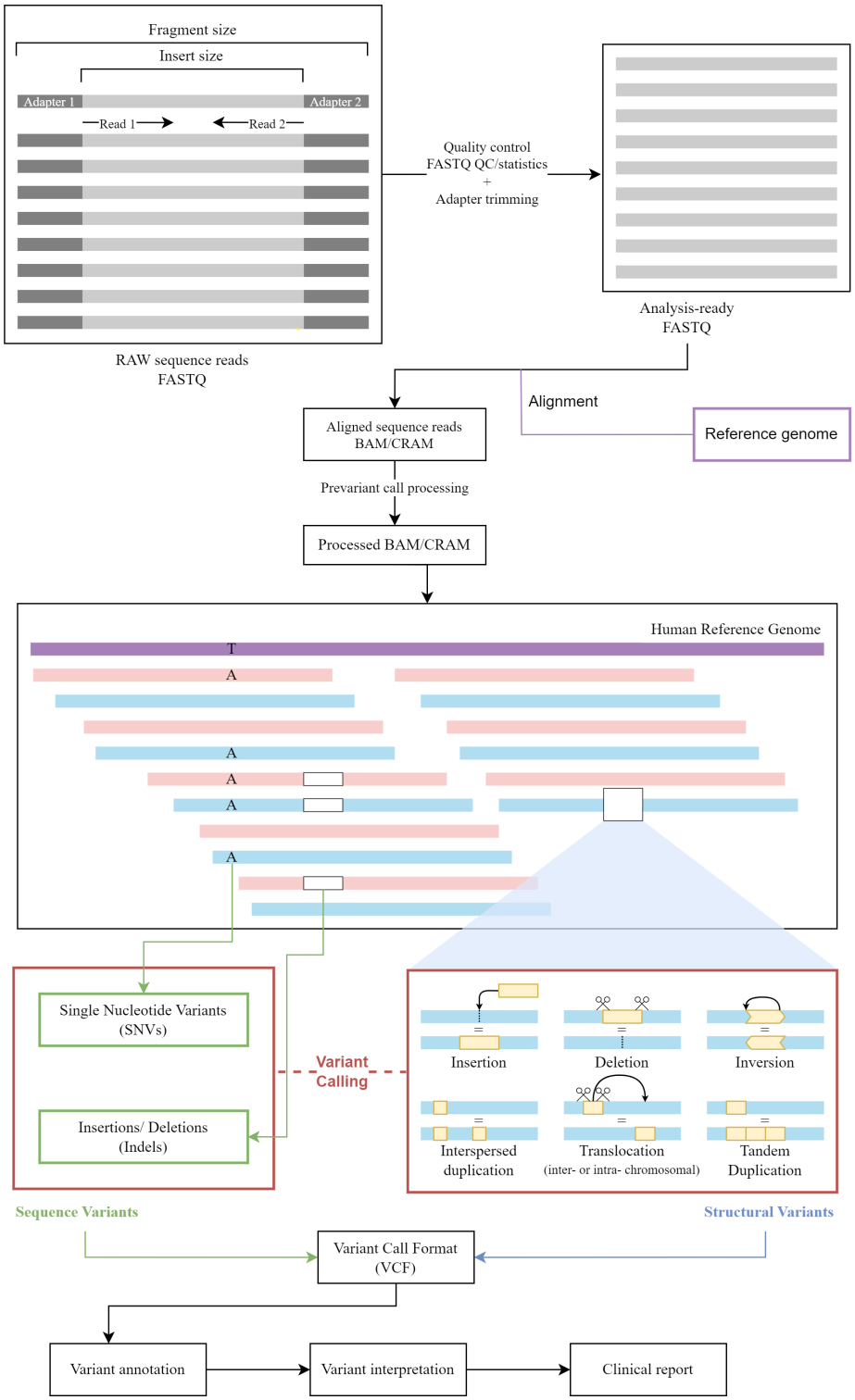


Figure 1.7: Overview of the bioinformatics pipeline for next-generation sequencing (NGS) analysis. DNA fragments with adapter sequences are processed to generate raw sequence reads (FASTQ). After quality control and adapter trimming, analysis-ready FASTQ files are aligned to a reference genome, producing aligned BAMCRAM files. These are further processed to identify sequence variants, including SNVs, INDELs, and structural variants such as translocations, duplications, and inversions. The identified variants are compiled in a VCF file, which undergoes annotation and interpretation to generate a clinical report. Adapted from [23], [24], [25], [26]

1.7.2 Next Generation Sequencing - Validation

In genetic and genomic analysis, namely in NGS, a key focus is often placed on the comparison between Coverage or Breadth of Coverage and Sequencing/Read Depth or Depth of Coverage. While these terms are related, they offer distinct insights into the quality and reliability of sequencing data. A thorough understanding of both concepts is essential to ensure accurate and meaningful results in genetic studies.

Coverage or Breadth of Coverage

Several elements could influence the total sequencing output, including the efficiency of the libraries, the extent of sample multiplexing, and the number of sequencing cycles. Given that the number of lanes in the flow cell is fixed, adjusting these factors is key to maximizing throughput. This throughput is commonly described in terms of Breadth of Coverage, which measures the proportion of the genome or target region that has been sequenced at least once. [21]

Breadth of Coverage focuses on ensuring that a substantial portion of the genome, or a specific target region such as the exome or a gene panel, has been adequately sequenced. It is typically expressed as a percentage. For instance, if 95% of the intended target region has been sequenced 30X, the Breadth of Coverage at 30X would be expressed as "95% Breadth of Coverage." However, several factors can influence Breadth of Coverage, including DNA sample quality, sequencing biases, and complexities within the genome, such as regions with high GC content or repetitive sequences, which can be challenging to sequence effectively. [38]

Sequencing/Read Depth or Depth of Coverage

Sequencing/Read Depth or Depth of Coverage, denotes the number of times a specific nucleotide in the DNA sequence is read during the sequencing process. It provides an indication of how thoroughly each base has been examined. The more times a nucleotide is read, the more confidence researchers can have in the accuracy of the base call, as a greater number of reads help mitigate the risk of sequencing errors and minimize noise. For instance, if a particular nucleotide is read 30 times, the Depth of Coverage at that location would be expressed as 30X. [38]

Higher Depth of Coverage means that more sequencing reads overlap each base, leading to improved sensitivity and accuracy in identifying variations. In contrast, lower Depth of Coverage can increase the capacity to sequence more samples or cover larger genomic areas at similar costs, although it might compromise the confidence in variant detection. [21]

Different types of sequencing require varying levels of Depth of Coverage. For Whole Genome Sequencing (WES), Depth of Coverage between 30X and 50X is generally recommended, while WES typically calls for 100X Depth of Coverage. In the case of targeted gene panels, the Depth of Coverage is often much greater-exceeding 500X in some cases-to ensure a high level of confidence in variant detection, particularly for somatic mutations, where greater read depth is crucial for accurate identification. [21]

The Figure 1.8 illustrates the relationship between Breadth of Coverage and Depth of Coverage in a gene. In this example, approximately 10% of the gene is depicted as not covered, and the Depth of Coverage reaches up to 9X. This visualization highlights the importance of both Breadth of Coverage and Depth of Coverage in ensuring the reliability of sequencing data.

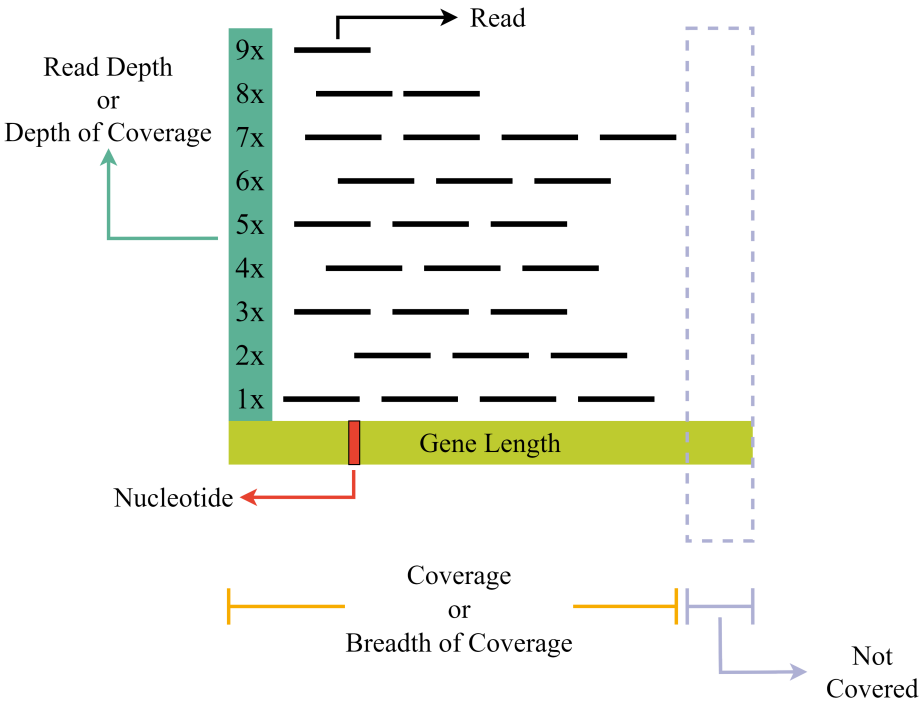


Figure 1.8: Scheme related to Coverage/Breadth of Coverage and Read Depth/Depth of Coverage in a gene. In this case, approximately 10% of the gene is depicted as not covered, and the depth reaches up to 9x. Adapted from [39]

Given the critical importance of validation in NGS, as previously demonstrated, this thesis focuses on the development of the following user-friendly software solution designed to calculate and ensure the accuracy of key metrics such as Breadth of Coverage and Depth of Coverage. These metrics are essential for evaluating the quality and reliability of sequencing data. The software presented here not only automates these calculations but also provides a streamlined and accessible interface, making it easier for users to perform NGS validation processes with consistency and precision.

CHAPTER 2

Software development process

"Don't worry about how you 'should' draw it. Just draw it the way you see it." - Tim Burton, filmmaker

2.1 Requirements

This section presents the requirements for the developed software, categorized into functional and non-functional requirements. The requirements were defined based on the needs of the company and end-users, ensuring that the software meets the expected performance, usability, and security standards.

2.1.1 Functional Requirements

Functional Requirements (FR) specify the functionalities that end-users require as essential components of the system. They describe the system's behavior in terms of inputs, operations to be performed, and expected outcomes. These requirements are directly observable by users in the final product. [40]

FR1 - Collection of BAM/CRAM Files for Analysis

The software must enable the collection of BAM/CRAM sequencing files stored on the company's servers.

FR2 - Calculation of Depth of Coverage/Read Depth and Coverage/Breadth of Coverage

The software must facilitate the calculation of Depth of Coverage and Breadth of Coverage for the collected BAM/CRAM files. Users should be able to configure analysis parameters, such as selecting regions of interest within the exome. The analysis should be based on a universal BED file containing genomic coordinates. The analysis should use bioinformatics tools like Samtools, which returns a DEPTH file with results that must be processed by the software to obtain the desired metrics.

FR3 - Graphical User Interface

The software must have a graphical user interface that allows users to interact with the system in an intuitive and efficient manner. The interface should be simple and easy to use, enabling users to collect BAM/CRAM files, configure analysis parameters, and view the obtained results. The interface should be developed using Streamlit, a Python web development tool that allows the creation of interactive web applications with minimal code. It should support user interaction through widgets such as buttons, text boxes, sliders, and others. The interface should allow filtering of results and exporting results to a Comma-Separated Values (CSV)/PDF file.

2.1.2 Non-Functional Requirements

Non-Functional Requirements (NFR) refer to the quality attributes of the software, such as performance, usability, security, and scalability. These requirements are crucial to ensure that the software operates efficiently, is secure, and can be easily used by various user profiles. [40] The main non-functional requirements of the system are described as follows:

NFR1 - Usability

The software should be intuitive and user-friendly, enabling even users with no technical background to interact with the system efficiently. The graphical user interface should be simple and clear, with straightforward instructions on how to use the system. It should allow users to collect BAM/CRAM files, configure analysis parameters, and view results quickly and easily. Clear and informative error messages should be provided in case of task execution failures, along with instructions for resolving issues.

NFR2 - Performance

The software must be optimized to process large volumes of sequencing data, ensuring that the calculation of Depth of Coverage/Read Depth and Coverage/Breadth of Coverage occurs within a reasonable time frame. The possibility of parallelizing calculations should be explored, utilizing multicore or distributed resources to accelerate data processing.

NFR3 - Scalability

The system must be scalable, capable of handling significant increases in data volume or the number of users without compromising performance. This includes the ability to leverage cloud services such as AWS S3. The software should be designed to allow the addition of new modules or functionalities without the need to rewrite the core code, ensuring flexibility for future evolution of the system.

NFR4 - Security and Data Privacy

The software must protect sensitive data, especially patient-related data, in compliance with regulations such as General Data Protection Regulation (GDPR). Temporary data used during processing must be properly deleted after analysis, ensuring data privacy and security. System access should be controlled through authentication and authorization, ensuring that

only authorized users can interact with the system. A logging system should be implemented to record user activities and monitor system usage.

NFR5 - Portability and Compatibility

The software should be implemented on Windows but must ensure access to a Linux environment using Windows Subsystem for Linux (WSL). It should guarantee easy integration with tools like Samtools in the WSL environment without requiring advanced configuration by the end user.

NFR6 - Maintainability

The software code must be modular and well-documented, facilitating easy maintenance and extension of the system in the future. Best development practices, such as version control (Git), should be applied, ensuring that the code can be managed efficiently over time. Software updates should be simple to implement, and developer documentation should include clear instructions on how to add new functionalities or adjust existing behaviors.

With clear definitions of FR and NFR, the development of the software was structured efficiently, ensuring it meets both technical expectations and operational needs of the company and end-users. Considering these requirements throughout the development cycle was crucial for the success of the project.

2.2 System Design and Architecture

The developed software is based on a web interface accessible through a browser, using the Streamlit library to build the application. The interface is composed of several interactive widgets that allow the user to configure and execute different types of genomic analysis: Single Gene, Gene Panel, or Exome.

2.2.1 User Workflow

Initially, the user must select the type of analysis to be performed. Depending on the selection, the processing flow adapts to optimize both the user experience and the efficiency of the necessary calculations.

Single Gene Analysis

The user selects a BAM or CRAM file, containing the sequencing data. Additionally, he automatically receive a universal BED file corresponding to the selected genome assembly. The user then chooses the gene of interest and may specify the exon(s) to be analyzed. Samtools is invoked to generate a DEPTH file, which contains information about the depth at each genomic position. This file is processed by a Python script that calculates the key metrics such as depth of coverage and breadth of coverage. The results are displayed to the user through a report in the Streamlit interface or can be exported as a CSV or PDF file.

Gene Panel Analysis

Similar to the Single Gene analysis, the user selects a BAM/CRAM file and the universal BED file. However, in this case, the user selects the Gene Panel for the analysis instead of a Single Gene. The corresponding BED file is processed by Samtools to generate the DEPTH file, which is then used to calculate the same coverage metrics for the whole panel, for each gene and for each exon of each gene. The result visualization and export process follow the same steps as in the Single Gene analysis.

Exome Analysis

In Exome analysis, the entire exome is used for the analysis, without requiring the selection of a specific region. The user selects the BAM/CRAM file, and the universal BED file corresponding to the exome. As in the other modes, Samtools generates the DEPTH file, which is processed to calculate the metrics. The results are displayed or exported in the same manner.

This modular design of the software allows for seamless integration between various components, namely Streamlit for the interface, Samtools for processing sequencing data, and Python scripts for calculating metrics. Each of these steps is interconnected to provide the user with quick, accurate, and real-time results. The software's architecture is designed to be easily scalable and adaptable to different types of genomic analysis, supporting both focused and large-scale analyses ES.

The Figure 2.1 shows the scheme of the software architecture, highlighting the main components.

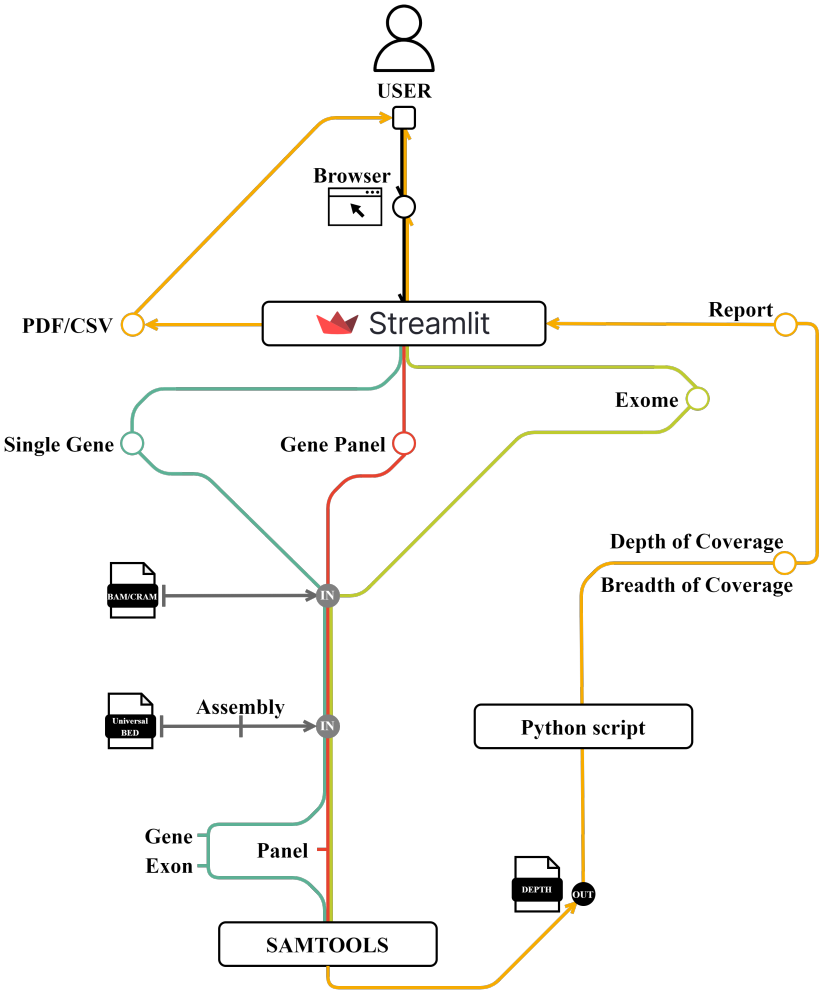


Figure 2.1: Scheme of the software architecture.

At the top of the scheme, Streamlit is positioned prominently allowing the user to select what analysis he wants to perform. The Streamlit interface is the central component of the system, symbolizing its role in managing the user interface and orchestrating the pipeline's execution. Streamlit receives input from several sources, including BAM/CRAM files and a Universal BED file.

These inputs are fed into a central processing unit, represented by Samtools, a powerful toolkit widely used for manipulating and analyzing high-throughput sequencing data.

From Samtools, a series of processing steps take place. The paths culminate in the generation of a DEPTH output file, which contains Depth of Coverage information for each genomic position analyzed. This DEPTH file is then fed into a Python script for further computation and analysis.

The final output from the Python script is routed back to the Streamlit application, completing the loop. The Streamlit interface is then responsible for displaying the results to the user, allowing them to interact with and visualize the processed data.

The scheme effectively captures the modularity of the system, with Samtools acting as the computational engine, Streamlit as the user interface, and Python handling the data

analysis. The diagram highlights the simplicity of the system's architecture while emphasizing the importance of data inputs, the computational process, and the feedback loop to the user interface.

The modularity of the system is demonstrated by the clear separation of components, each with a specific role in the analysis process. The Figure 2.2 shows the software directory structure.

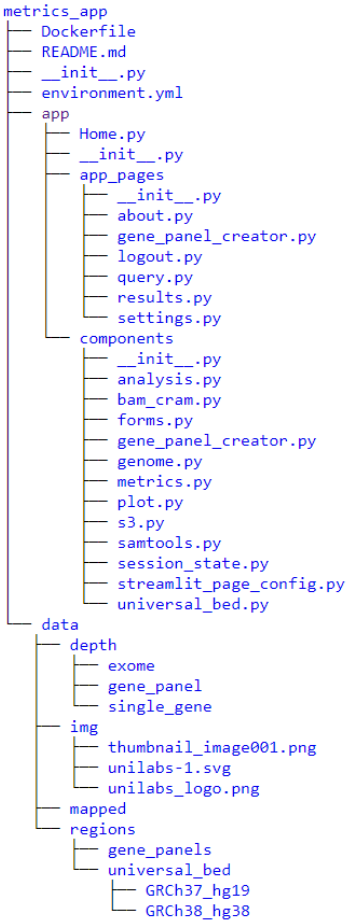


Figure 2.2: Software directory structure.

At the top level, the `metrics_app` directory contains core components such as the app logic and necessary resources. The `app` folder houses the main scripts that drive the application, including `Home.py`, which serves as the entry point, and submodules like `app_pages` for UI-specific components such as `about.py`, `query.py`, and `results.py`. The `components` directory contains functional modules such as `analysis.py`, `metrics.py`, and `bam_cram.py`, encapsulating core logic for genome analysis, metrics computation, and file handling. The `data` folder stores essential files, including genomic depth and region data, which are organized by specific analysis types (e.g., Single Gene, Gene Panel). The project includes setup files like `environment.yml` for managing dependencies and a `Dockerfile` to enable containerization for consistent deployment. Overall, the structure supports scalability, modularity, and ease of maintenance.

2.3 Development

The development of the software followed an iterative and structured approach, with each stage focusing on expanding its functionality while ensuring stability and performance. The process began with the implementation of the Single Gene analysis, which served as the foundation for the project. This initial version was designed to be stable and functional, enabling users to select BAM or CRAM files and analyze metrics for specific genes with precision.

Once the Single Gene functionality was fully operational, the next phase involved adding support for Gene Panel analysis. This required adjusting the existing framework to handle a broader set of genes while maintaining the same level of accuracy and efficiency. A stable and functional version was again achieved, providing users with the ability to select and analyze predefined gene panels. Each functionality added was versioned using Git and GitHub, ensuring that changes were tracked and managed effectively.

Finally, the software was further enhanced to include Exome analysis, allowing for the comprehensive examination of the entire exome. This functionality was integrated without compromising the stability or performance of the system, and once again, a stable and functional version was built by adding a branch to project's GitHub repository.

Throughout the development, various components and tools were employed to ensure the software met the required performance standards and provided an intuitive user experience. The careful progression from Single Gene to Gene Panel, and finally to Exome analysis, highlights the modularity and scalability of the software, as well as the emphasis placed on testing and validation at each stage.

The following sections provide an overview of the key tools and technologies used in the development of the whole software, including all the stages of the project.

2.3.1 Environment preparation

Windows Subsystem for Linux

On Windows, developers have access to both the Windows and Linux environments, thanks to the WSL. With WSL, it is possible to install different Linux distributions, such as Ubuntu, OpenSUSE, Kali, Debian, Arch Linux, among others. This allows Linux applications, utilities and command-line tools to be used directly in Windows, without the need to modify the operating system, resort to virtual machines or dual boot. [41]

In the context of the development of this tool, the need to install WSL was driven mainly due to the scenario in which many essential tools and software for bioinformatics are designed to work in Linux environments. For this reason, the set of steps recommended on the Microsoft website to configure this environment (Build 19041 or higher) were followed. [41]

Anaconda and Conda

After installing WSL, the installation step of Anaconda (a platform for data science in Python/R that includes conda, a package and environment manager) followed. [42]

In the case of the creation of the metrics analysis tool, this step was fundamental to allow the installation and maintenance of all the packages and dependencies necessary for the operation of the software. By creating a conda environment, it was possible to ensure that all installed tools work independently without conflicts between versions and packages, thus ensuring the reproducibility of the created software. [43]

Following the documentation provided by Anaconda, the installation and creation of the conda environment was carried out. [44]

Additionally, all the dependencies of the Table A.3 were installed within the created environment. This installation was carried out by installing package by package, however, an environment.yaml file was made available, allowing the bulk installation [45] of all dependencies on the versions compatible with the software.

Git and GitHub

GitHub is a platform that allows users to store, share, and collaborate on code writing with others. [46] Its operation is based on repositories managed by Git, a version control system that tracks all changes made by one or more users in a project. [46] When files are uploaded to GitHub, they become part of the created repository. Any change (commit) to any file is automatically tracked. These changes, made locally, are usually synchronized continuously by pushing the committed changes. Similarly, any changes made locally by another user and synchronized on GitHub can be retrieved by making a pull request. [46]

Thus, by using the documentation of Git and GitHub, this practice was implemented, which not only ensures that each version of the created software is recorded, guaranteeing that the work is not lost and allowing for version rollback in case of bugs, but also ensures that all software produced is reproducible and available for deployment by any user. [46]

2.3.2 Streamlit

The developed software was built using Streamlit which is an open-source Python library designed to enable the swift and intuitive creation of interactive web applications, primarily aimed at data science and machine learning projects. Introduced in October 2019 and now part of the Snowflake ecosystem, Streamlit has quickly gained recognition within the data science community due to its user-friendly approach. Its main goal is to make transforming machine learning scripts into interactive apps as simple as possible, allowing users to incorporate complex features with minimal effort. [47]

Streamlit's core philosophy revolves around a declarative, straightforward interface-building model. It enables developers to create apps using clean, concise code without the need for managing intricate state or setting up callbacks, which is common in other frameworks. This makes Streamlit especially suited for rapid prototyping, as well as for displaying machine learning models and data visualizations. The library also seamlessly integrates with many popular Python frameworks for data analysis and visualization. [47]

What further sets Streamlit apart is its flexibility and extensibility. The library's architecture allows the integration of various web components, making it easy to customize applications for different use cases. This versatility, along with its growing popularity and

adoption in the data science world, has cemented Streamlit as a valuable tool for building interactive, informative applications that can be easily shared and adapted. [48]

Various Streamlit widgets were used to create the graphical interface of the software, including buttons, text boxes, selectors, among others. Additionally, the library was employed to display the analysis results, also allowing data export to a CSV/PDF file. The integration of Streamlit with Python was crucial in developing an interactive and user-friendly software, meeting the established usability requirements. The functionality and implementation of these widgets are detailed in [49].

2.3.3 Universal BED - Processing and Simplification of BED Files for Genomic Analysis and Integration of MANE Transcripts

The analysis of genomic data often requires simplifying complex genomic annotation files while ensuring that all relevant information is retained. In this work, BED files were obtained using Ensembl's BioMart [50] tool, specifically by querying the MANE Select and MANE Plus Clinical [51] transcripts for the GRCh38 genome version, and Ensembl annotations for GRCh37. Since there are no MANE transcripts available for GRCh37, the MANE equivalence was determined by leveraging the Ensembl TARK [52] database, which provides a list of transcripts from GRCh37 that match in 5' UTR, 3' UTR, and CDS with MANE Select and MANE Plus Clinical transcripts on GRCh38. The matching was facilitated by using an auxiliary CSV file.

The BED files retrieved were pre-simplified using specific BioMart queries, which focused on selecting only key columns that are critical for our genomic analysis, namely chromosome name, exon coordinates, gene stable identifier, gene name, exon ID, strand, and transcript ID. This initial pre-processing helped reduce the size of the BED files and provided more straightforward data to work with. The following URLs represent the BioMart queries used to generate the BED files:

- MANE Select GRCh38
- MANE Plus Clinical GRCh38
- Ensembl GRCh37

The importance of using MANE transcripts cannot be understated. The MANE project, a collaborative effort by NCBI and Ensembl, aims to provide a single, matched transcript for each human protein-coding gene, ensuring a standardized reference across major databases. This integration was crucial for harmonizing the annotations and minimizing discrepancies that may arise from different transcript versions. By focusing on MANE Select and MANE Plus Clinical transcripts, the software ensures consistency and reliability in metrics calculation, which is particularly important for both research and clinical genomic analyses.

Data Transformation Process

To convert the BioMart output files into a more workable form, a two-step process was followed involving the use of custom Python scripts that can be found in the folder `tools` of the repository. First, the original BED files were converted into JSON format using the

script presented below. This step was necessary to facilitate easier parsing and handling of gene-specific information.

```python
import json
import argparse

def bed_to_json(input_file, output_file):
    bed_records = []

    with open(input_file, 'r') as infile:
        # Ignore the header line
        header = infile.readline()

        for line in infile:
            # Ignore additional header lines or empty lines
            if line.startswith("#") or line.strip() == "":
                continue

            fields = line.strip().split('\t')

            # Adjust according to the expected BED format (in this case, 8 columns)
            if len(fields) >= 8:
                chromosome = fields[0]
                start = int(fields[1])
                end = int(fields[2])
                gene_stable_id = fields[3]
                gene_name = fields[4]
                exon_id = fields[5]
                strand = fields[6]
                transcript_id = fields[7]

                bed_records.append({
                    "chromosome": chromosome,
                    "start": start,
                    "end": end,
                    "gene_stable_id": gene_stable_id,
                    "gene_name": gene_name,
                    "exon_id": exon_id,
                    "strand": strand,
                    "transcript_id": transcript_id
                })

    with open(output_file, 'w') as jsonfile:
        json.dump(bed_records, jsonfile, indent=4)


def main():
    parser = argparse.ArgumentParser(description="Convert a BED file to JSON.")
    parser.add_argument('input_file', help="Path to the input BED file.")
    parser.add_argument('output_file', help="Path to the output JSON file.")
```

```
    args = parser.parse_args()

    bed_to_json(args.input_file, args.output_file)

if __name__ == "__main__":
    main()
```

Code 1: Python script for converting BED files to JSON format.

This script (`bed_to_json.py`) reads the input BED file, processes each line, and writes the relevant fields into a structured JSON file. By ignoring the header and retaining only key fields, the script produces a lightweight and usable JSON representation of the BED data.

Processing the JSON Files

Once the JSON files were obtained, the next step involved processing these files to generate a unified and simplified BED file for GRCh37 and for GRCh38 genome assemblies. The script `universal_bed.py` achieves this, as described in the following code snippets.

The first step is loading the necessary mappings for gene identifiers and MANE equivalence. This helps ensure that GRCh37 annotations are harmonized with GRCh38, as shown in Code 2.

```
import json
import csv


def load_gene_mapping(mapping_file):
    with open(mapping_file, 'r') as json_file:
        return json.load(json_file)


def load_mane_mapping(mane_csv):
    mane_mapping = {}
    with open(mane_csv, 'r') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            gene_name = row['Gene']
            grch37_id = row['Ensembl StableID GRCh37 (Not MANE)']
            if grch37_id != '':
                mane_mapping[grch37_id] = gene_name
    return mane_mapping
```

Code 2: Loading mappings for gene identifiers and MANE equivalence.

The `load_gene_mapping()` function loads a gene mapping from a JSON file, while `load_mane_mapping()` loads MANE transcript mappings from a CSV file. These mappings are used to reconcile differences between GRCh37 and GRCh38 annotations.

The main function of the script reads the input JSON files and processes the genomic data accordingly. The Code 3 shows the core of the function that performs data transformation.

```python
def process_bed_file(input_files, output_file, genome_version, mapping_file=None,
↪  mane_file=None, add_chr_prefix=False):
    gene_mapping = load_gene_mapping(mapping_file) if mapping_file else {}
    mane_mapping = load_mane_mapping(mane_file) if mane_file else {}

    exon_counts = {}
    rows = []

    for input_file in input_files:
        with open(input_file, 'r') as infile:
            bed_data = json.load(infile)
            for record in bed_data:
                chromosome = record['chromosome']
                start = record['start']
                end = record['end']
                gene_id = record['gene_stable_id']
                gene_name = record.get('gene_name', '')
                exon_id = record['exon_id']
                strand = record['strand']
                transcript_id = record['transcript_id']

                # Only include standard chromosomes (1-22, X, Y)
                if not re.match(r'^(\d+|X|Y)$', chromosome):
                    continue

                # Apply MANE filtering for hg37
                if genome_version == 'hg37' and mane_mapping and transcript_id not in
                ↪  mane_mapping:
                    continue
                elif genome_version == 'hg37' and transcript_id in mane_mapping:
                    gene_name = mane_mapping[transcript_id]

                # Add 'chr' prefix if needed
                if add_chr_prefix:
                    chromosome = f'chr{chromosome}'

                rows.append((chromosome, int(start), int(end), gene_name, strand))
```

Code 3: Processing JSON files to generate a simplified BED file.

This function reads each record from the JSON files, applies necessary filters (such as selecting standard chromosomes and applying MANE mapping for GRCh37), and constructs rows for the final BED output. Depending on the user's choice, the chromosome names can be prefixed with "chr".

In the final step, the script assigns exon numbers and calculates exon lengths (Code 4).

```python
    # Assign exon numbers
    for (gene_name, strand), exon_list in exon_counts.items():
        if strand == '1':
```

```
            for exon_number, (chromosome, start, end, gene_name, strand) in
            ↪ enumerate(exon_list, start=1):
                exon_length = end - start
                updated_rows.append((chromosome, start, end, gene_name, exon_number,
                ↪ exon_length, strand))
        else:
            for exon_number, (chromosome, start, end, gene_name, strand) in
            ↪ enumerate(exon_list, start=1):
                exon_length = end - start
                updated_rows.append((chromosome, start, end, gene_name, len(exon_list) -
                ↪ exon_number + 1, exon_length, strand))

    rows = updated_rows

    # Write to output file
    with open(output_file, 'w') as outfile:
        for row in rows:
            outfile.write(f"{row[0]}\t{row[1]}\t{row[2]}\t{row[3]}\t{row[4]}\t{row[5]}\t{r
            ↪ ow[6]}\n")
```

Code 4: Assigning exon numbers and writing to output BED file.

This code snippet assigns exon numbers based on the strand orientation, calculates exon lengths, and writes the final processed data to an output BED file. The careful assignment of exon numbers ensures that the output accurately reflects the genomic structure, which is essential for subsequent gene and exon-level analysis.

The processing and simplification of BED files, coupled with the integration of MANE transcripts, ensured that the genomic data was standardized and simplified to facilitate efficient and accurate downstream analysis. The use of MANE transcripts, in particular, helped minimize discrepancies across different data sources, contributing to the consistency and reliability of key metrics such as Depth of Coverage and Breadth of Coverage. By using a well-defined data transformation process, the software's analysis capabilities were enhanced, making it more suitable for both research and clinical genomic applications.

2.3.4 Samtools Depth Calculation in Python with Streamlit

The **depth** function from Code 5 is designed to compute the Depth of Coverage for specific genes and exons from CRAM or BAM sequencing files. This function integrates Samtools into a Python workflow, allowing users to filter genomic regions dynamically based on the provided genes and exons. By leveraging Streamlit's session state, it retains filtered BED content and the depth output for subsequent processing and visualization.

```
def depth(cram_path, bed_path, depth_dir='data/depth', gene_selection=None,
↪ exon_selection=None):
    """
    Calculate the depth of coverage for specific exons of genes in a CRAM/BAM file using
    ↪ samtools.

```

```
    Args:
        cram_path (str): Path to the CRAM/BAM file.
        bed_path (str): Path to the Universal BED file containing exon coordinates.
        depth_dir (str, optional): Directory to save the depth output file (optional if
        ↪  storing in session state).
        gene_selection (list or None): List of gene names to include in the depth
        ↪  calculation.
        exon_selection (list or None): List of exon numbers to include in the depth
        ↪  calculation for each gene.

    Returns:
        None
    """
```

Code 5: Python function for calculating Depth of Coverage using Samtools.

Initial Setup and Session State Initialization

The function starts by initializing two important variables within the Streamlit session state: `depth_output` and `filtered_bed`. These variables will hold, respectively, the Samtools depth output and the filtered BED content after processing (Code 6). By storing these variables in session state, the function ensures that intermediate results can persist across different user interactions.

```
# Initialize session state attributes if they don't exist
if 'depth_output' not in st.session_state:
    st.session_state.depth_output = {}

if 'filtered_bed' not in st.session_state:
    st.session_state.filtered_bed = ""
```

Code 6: Session state initialization.

File Validation and Setup for Gene and Exon Selection

Before executing the core logic, the function verifies that the paths provided for the BAM/CRAM and BED files are valid (Code 7). This validation ensures that no downstream operations occur with invalid input files, avoiding potential errors during execution. If the provided file paths are incorrect, the function halts, displaying an error message via Streamlit's `st.error` function.

In cases where specific genes or exons are provided, the function converts these selections into lists, ensuring proper format handling. If no gene or exon selection is provided, the function defaults to reading the original BED file and storing its contents in the session state. This setup supports two modes of operation: calculating depth across the entire BED file or focusing on specific regions of interest.

```python
# Check if paths to the CRAM/BAM and BED files are valid
if not os.path.isfile(cram_path) or not os.path.isfile(bed_path):
    st.error("Invalid file path provided for CRAM/BAM or BED file.")
    return  # Stop execution if the paths are invalid

# Convert gene_selection to a list if it's a string
if isinstance(gene_selection, str):
    gene_selection = [gene_selection]

# Convert exon_selection to a list of strings if it's a list of numbers
if exon_selection is not None:
    exon_selection = list(map(str, exon_selection))  # Convert exon numbers to strings

# If no gene or exon selection is provided, use the original BED file
if gene_selection is None and exon_selection is None:
    try:
        with open(bed_path, 'r') as bed_file:
            st.session_state.filtered_bed = bed_file.read()  # Store original BED content
            ↪ in session state
    except Exception as e:
        st.error(f"Error loading BED file: {e}")
        return  # Stop execution in case of file read error
```

Code 7: Validation and setup for gene and exon selection.

Filtering the BED File Based on Gene and Exon Selections

When specific genes or exons are provided, the function dynamically filters the BED file to isolate only the relevant regions for analysis. This filtering process iterates through the lines of the BED file, matching the genes and exons with the user-provided selections. Any lines that match the criteria are retained and stored in the session state (Code 8).

This step employs a `try-except` block to handle potential file processing errors gracefully. If no regions match the provided criteria, an informational message is displayed to the user, and the function exits early. This dynamic filtering enhances the function's flexibility by allowing users to focus on specific genomic intervals, ensuring that only relevant data is processed in subsequent steps.

```python
else:
    # Otherwise, filter the BED file based on the gene and exon selection
    filtered_bed_lines = []
    try:
        with open(bed_path, 'r') as bed_file:
            for line in bed_file:
                columns = line.strip().split('\t')

                # Ensure the BED file has at least 6 columns (chr, start, end, gene, exon,
                ↪ size)
                if len(columns) < 6:
```

```
                    continue

                chrom, start, end, gene, exon, size = columns[0], columns[1], columns[2],
                ↪  columns[3], columns[4], columns[5]

                # Apply gene filter (multiple genes allowed)
                if gene_selection is None or gene in gene_selection:
                    # Apply exon filter: if exon_selection is None, include all exons for
                    ↪  the gene
                    if exon_selection is None or exon in exon_selection:
                        filtered_bed_lines.append(f"{chrom}\t{start}\t{end}\t{gene}\t{exon
                        ↪  }\t{size}\n")

        # Check if any filtered BED content was found
        if not filtered_bed_lines:
            st.info("No matching regions found for the provided gene or exon selection.")
            return  # Stop execution if no matching regions found

        # Store filtered BED content in Streamlit session state
        st.session_state.filtered_bed = ''.join(filtered_bed_lines)

    except Exception as e:
        st.error(f"Error filtering BED file: {e}")
        return  # Stop execution in case of file processing error
```

Code 8: Filtering the BED file based on gene and exon selections.

Running Samtools and Handling the Depth Output

Once the filtered regions are prepared, the function proceeds to run Samtools `depth` command. This step is also encapsulated in a `try-except` block to handle potential errors during Samtools execution (Code 9). The filtered BED content, stored in session state, is passed to Samtools via standard input, allowing the command to compute the Depth of Coverage for the specified regions.

After the Samtools process completes, the function checks whether any depth data was returned. If no data is produced (for example, if no coverage was found in the specified regions), the function exits. If data is present, the depth output is stored in the session state for further use. Optionally, the depth output can also be saved to a file in a specified directory, ensuring that users have access to a persistent output.

```
# Create a unique key for the output based on the CRAM/BAM file name
output_key = os.path.splitext(os.path.basename(cram_path))[0]

# Run samtools depth using the BED content (either original or filtered) from session state
try:
    samtools_command = ['samtools', 'depth', '-b', '-', cram_path]
    samtools_process = subprocess.Popen(samtools_command, stdin=subprocess.PIPE,
    ↪  stdout=subprocess.PIPE, text=True)
```

```
    samtools_output, _ = samtools_process.communicate(input=st.session_state.filtered_bed)

    # Check if samtools_output is empty
    if not samtools_output.strip():  # If no output is found
        return  # Stop execution if no data is returned

    # Store samtools output (.depth content) in Streamlit session state as a dictionary
    st.session_state.depth_output[output_key] = samtools_output

    # Optionally, save the depth data to a file
    if depth_dir:
        # Ensure the directory exists
        os.makedirs(depth_dir, exist_ok=True)

        # Define the output depth file path based on the CRAM/BAM file name
        depth_path = os.path.join(depth_dir, f"{output_key}.depth")
        with open(depth_path, 'w') as output_file:
            output_file.write(samtools_output)

except Exception as e:
    st.error(f"Error running samtools depth: {e}")
    return  # Stop execution in case of samtools error
```

Code 9: Running Samtools and handling depth output.

Example Usage of the Depth Function

Code 10 provides an example usage of the **depth** function. In this scenario, the function is used to compute the depth of coverage for the **BRCA1** gene across exons 1, 2, and 3 from a given CRAM file. The depth output is stored in a specified directory, ensuring that the results can be accessed for further analysis.

```
depth(
    cram_path="/path/to/sample.cram",
    bed_path="/path/to/regions.bed",
    depth_dir="/path/to/output_dir",
    gene_selection=["BRCA1"],
    exon_selection=[1, 2, 3]
)
```

Code 10: Example usage of the depth function.

2.3.5 Detailed Breakdown of the Python Script for Metrics Calculation

This section delves into a Python script designed to compute sequencing coverage metrics from DEPTH file outputed by Samtools. The script operates within a Streamlit session, facilitating dynamic filtering of genomic regions and providing statistical summaries for individual genes and exons. Below, it's explored each component of the script in detail, illustrating how it achieves its goal of efficient metrics calculation.

Importing Necessary Libraries

The script begins by importing essential Python libraries required for data manipulation and interaction with the Streamlit application. The core libraries include `pandas`, a library that provides robust data structures, such as DataFrames, which facilitate complex data manipulation and analysis, `io` a module that allows the script to handle in-memory streams, which are particularly useful for reading and processing textual data from files stored in memory and, finally, `streamlit`, the package who enables the creation of the user interface.

```
import pandas as pd
import io
import streamlit as st
```

Code 11: Importing necessary libraries.

Defining the Metrics Initialization Function

Next, the script defines the `initialize_metrics()` function, which is responsible for setting up a consistent data structure for storing various sequencing metrics. This function returns a dictionary where each key corresponds to a specific metric, and all metrics are initialized to zero. This standardization ensures that metrics are consistently tracked and updated throughout the calculation process.

```
def initialize_metrics():
    return {
        'Size Coding': 0,
        'Size Covered': 0,
        'Breadth of Coverage %': 0,
        'Average Read Depth': 0,
        'Min Read Depth': 0,
        'Max Read Depth': 0,
        'Depth of Coverage % (1x)': 0,
        'Depth of Coverage % (10x)': 0,
        'Depth of Coverage % (15x)': 0,
        'Depth of Coverage % (20x)': 0,
        'Depth of Coverage % (30x)': 0,
        'Depth of Coverage % (50x)': 0,
        'Depth of Coverage % (100x)': 0,
        'Depth of Coverage % (500x)': 0,
        'Depth of Coverage (>500x)': 0,
        'Depth of Coverage (0-1x)': 0,
        'Depth of Coverage (2-10x)': 0,
        'Depth of Coverage (11-15x)': 0,
        'Depth of Coverage (16-20x)': 0,
        'Depth of Coverage (21-30x)': 0,
        'Depth of Coverage (31-50x)': 0,
        'Depth of Coverage (51-100x)': 0,
        'Depth of Coverage (101-500x)': 0
    }
```

Code 12: Defining the metrics initialization function.

This dictionary structure is critical for the script because it ensures that all necessary metrics, such as Depth of Coverage percentages at various thresholds, are included and consistently tracked. By initializing metrics to zero, the function guarantees that all relevant data fields are present, even if no coverage is detected in certain regions, thereby preventing errors later in the script.

Accessing Filtered Data from Session State

The `calculate_metrics()` function, which handles the core logic of the script, begins by accessing the filtered BED content and depth data that were stored in Streamlit's session state during earlier steps. These data sources are crucial for the downstream calculations. If either the BED content or the depth data is missing, the function raises a `ValueError`, ensuring that no further calculations are performed with incomplete or missing data.

```
def calculate_metrics():
    # Access the filtered BED content and depth data from session_state
    bed_content = st.session_state.get('filtered_bed', '')
    depth_dict = st.session_state.get('depth_output', {})

    if not bed_content:
        raise ValueError("No filtered BED content found in session state.")

    if not depth_dict:
        raise ValueError("No depth data found in session state.")

    results = {}  # Initialize results dictionary
```

Code 13: Accessing filtered BED and depth data from session state.

This step of accessing session state data ensures that the script can efficiently handle user-selected regions of interest (genes and exons or gene panels) and corresponding depth data. This setup also supports interactive workflows where users can refine their selections dynamically, with metrics recalculated based on these selections.

Reading the Filtered BED Content

The filtered BED content is then read into a Pandas DataFrame for easier manipulation. This DataFrame contains information about the genomic regions (chromosome, start and end positions, gene names, exon numbers, and exon sizes), which will later be cross-referenced with the depth data to compute metrics.

```
    # Read the filtered BED content into a DataFrame
    bed_df = pd.read_csv(io.StringIO(bed_content), sep='\t', header=None,
                         names=['CHROM', 'START', 'END', 'GENE', 'EXON', 'SIZE'])
```

Code 14: Reading filtered BED content into a DataFrame.

By converting the BED content into a DataFrame, the script leverages Pandas powerful data manipulation capabilities, which streamline tasks such as filtering, grouping, and aggregating genomic regions. This enables efficient computation of metrics for different genes and exons in subsequent steps.

Processing Depth Data for Each File

Next, the script iterates over each entry in the depth dictionary, where each entry corresponds to a different BAM or CRAM file. For each file, a new DataFrame is created to hold the depth data, which consists of three columns: chromosome (`CHROM`), position (`POS`), and depth (`DEPTH`).

```
    for file_name, depth_content in depth_dict.items():
        # Initialize the results structure for this file
        results[file_name] = {}

        # Read the depth content into a DataFrame
        depth_df = pd.read_csv(io.StringIO(depth_content), sep='\t', header=None,
                               names=['CHROM', 'POS', 'DEPTH'])
```

Code 15: Processing depth data for each BAM/CRAM file.

This iterative approach allows the script to handle multiple samples or files, computing metrics independently for each. This flexibility makes the script suitable for batch processing of sequencing data or for analyzing different samples within the same workflow.

Calculating Metrics for All Genes Combined

The script then calculates metrics for all genes combined, which provides an overall summary of the metrics across the entire set of regions. Key metrics include the total size of coding regions (`Size Coding`), the Average Depth of Coverage (`Average Read Depth`), the Breadth of Coverage.

```
        # Initialize metrics for all genes
        all_genes_metrics = initialize_metrics()
        total_positions = len(depth_df)

        if total_positions > 0:
            # Extract depth values
            all_depths = depth_df['DEPTH']

            # Calculate Size Coding
            all_genes_metrics['Size Coding'] = bed_df['SIZE'].sum()

            # Calculate Average Read Depth
            all_genes_metrics['Average Read Depth'] = all_depths.mean()

            # Calculate Min and Max Read Depth
            all
```

```

_genes_metrics['Min Read Depth'] = all_depths.min()
            all_genes_metrics['Max Read Depth'] = all_depths.max()

            # Calculate Size Covered
            all_genes_metrics['Size Covered'] = all_depths.count()

            # Calculate Breadth of Coverage %
            all_genes_metrics['Breadth of Coverage %'] = (
                all_genes_metrics['Size Covered'] / all_genes_metrics['Size Coding']
            ) * 100
```

Code 16: Calculating metrics for all genes combined.

The calculation of `Breadth of Coverage %` is particularly important for assessing the quality of sequencing coverage, as it indicates the proportion of the target regions that have been successfully sequenced. These metrics provide a high-level overview of the data, serving as a starting point for more granular analysis at the gene and exon levels.

Calculating Depth of Coverage Percentages

For each depth threshold (e.g., 1x, 10x, 15x, etc.), the script computes the percentage of positions that meet or exceed that depth. These depth thresholds are commonly used in sequencing to assess the reliability of variant calls or to ensure sufficient coverage for accurate analysis.

```
            # Define coverage thresholds
            coverage_thresholds = [1, 10, 15, 20, 30, 50, 100, 500]

            # Calculate counts for each threshold
            coverage_counts = {
                threshold: (all_depths >= threshold).sum() for threshold in
                ↪  coverage_thresholds
            }

            # Calculate percentages
            for threshold in coverage_thresholds:
                all_genes_metrics[f"Depth of Coverage % ({threshold}x)"] = (
                    coverage_counts[threshold] / total_positions
                ) * 100
```

Code 17: Calculating depth of coverage percentages for different thresholds.

By calculating these percentages, the script provides insight into how well each region is covered at different depths, helping to identify any coverage gaps that might impact downstream analysis. These metrics are critical for quality control and for determining whether additional sequencing is needed.

Calculating Depth Intervals

In addition to calculating Depth of Coverage percentages for fixed thresholds, the script also computes the number of positions that fall within specific depth intervals. This helps to understand the distribution of Depth of Coverage across the different thresholds.

```
            # Define depth intervals and calculate counts
            depth_intervals = {
                "Depth of Coverage (>500x)": (all_depths > 500).sum(),
                "Depth of Coverage (101-500x)": ((all_depths > 100) & (all_depths <=
                ↪  500)).sum(),
                "Depth of Coverage (51-100x)": ((all_depths >= 51) & (all_depths <=
                ↪  100)).sum(),
                "Depth of Coverage (31-50x)": ((all_depths >= 31) & (all_depths <=
                ↪  50)).sum(),
                "Depth of Coverage (21-30x)": ((all_depths >= 21) & (all_depths <=
                ↪  30)).sum(),
                "Depth of Coverage (16-20x)": ((all_depths >= 16) & (all_depths <=
                ↪  20)).sum(),
                "Depth of Coverage (11-15x)": ((all_depths >= 11) & (all_depths <=
                ↪  15)).sum(),
                "Depth of Coverage (2-10x)": ((all_depths >= 2) & (all_depths <= 10)).sum(),
                "Depth of Coverage (0-1x)": (all_depths <= 1).sum(),
            }

            # Update the metrics dictionary
            all_genes_metrics.update(depth_intervals)
```

Code 18: Calculating counts for specific depth intervals.

Calculating Metrics for Each Gene

The script proceeds to calculate individual metrics for each gene. It filters the BED DataFrame to extract the corresponding regions, and then accumulates depth values across all positions in those regions. Metrics such as Average Read Depth, Minimum and Maximum Read Depth are calculated for each gene.

```
            # Initialize dictionaries to hold gene and exon metrics
            genes_data = {}
            exons_data = {}
            genes = bed_df['GENE'].unique()

            for gene in genes:
                # Initialize metrics for the gene
                gene_metrics = initialize_metrics()
                gene_bed_df = bed_df[bed_df['GENE'] == gene]

                # Calculate Size Coding for the gene
                gene_metrics['Size Coding'] = gene_bed_df['SIZE'].sum()

```

```
                # Initialize a Series to hold depth values for the gene
                gene_depths = pd.Series(dtype=float)

                # Accumulate depth values for all exons in the gene
                for _, row in gene_bed_df.iterrows():
                    start = row['START']
                    end = row['END']
                    exon_depths = depth_df[
                        (depth_df['POS'] >= start) & (depth_df['POS'] <= end)
                    ]['DEPTH']
                    gene_depths = pd.concat([gene_depths, exon_depths], ignore_index=True)

                if len(gene_depths) > 0:
                    # Calculate metrics for the gene
                    gene_metrics['Average Read Depth'] = gene_depths.mean()
                    gene_metrics['Min Read Depth'] = gene_depths.min()
                    gene_metrics['Max Read Depth'] = gene_depths.max()
                    gene_metrics['Size Covered'] = gene_depths.count()
                    gene_metrics['Breadth of Coverage %'] = (
                        gene_metrics['Size Covered'] / gene_metrics['Size Coding']
                    ) * 100
                else:
                    # Set metrics to zero if no depth data is available
                    gene_metrics['Average Read Depth'] = 0
                    gene_metrics['Min Read Depth'] = 0
                    gene_metrics['Max Read Depth'] = 0
                    gene_metrics['Size Covered'] = 0
```

Code 19: Calculating metrics for each gene.

This gene-specific analysis allows the script to provide detailed metrics that are relevant for studying individual genes. These metrics can be used to assess whether particular genes have been sequenced sufficiently for downstream analyses, such as variant calling or expression analysis.

Calculating Depth of Coverage Percentages for Each Gene

As with the overall analysis, the script calculates also Depth of Coverage percentages for each gene at various thresholds. This ensures that the analysis can identify specific genes that may have insufficient coverage for reliable variant calling or other analyses.

```
                # Calculate coverage percentages for the gene
                coverage_counts_gene = {
                    threshold: (gene_depths >= threshold).sum() for threshold in
                    ↪  coverage_thresholds
                }

                for threshold in coverage_thresholds:
                    gene_metrics[f"Depth of Coverage % ({threshold}x)"] = (
                        coverage_counts_gene[threshold] / gene_metrics['Size Covered']
```

```
            ) * 100 if gene_metrics['Size Covered'] > 0 else 0

        # Calculate depth intervals for the gene
        depth_intervals_gene = {
            "Depth of Coverage (>500x)": (gene_depths > 500).sum(),
            "Depth of Coverage (101-500x)": ((gene_depths > 100) & (gene_depths <=
            ↪  500)).sum(),
            # ... (other intervals)
        }

        gene_metrics.update(depth_intervals_gene)

        # Store the gene metrics
        genes_data[gene] = gene_metrics
```

Code 20: Calculating depth percentages and intervals for each gene.

Calculating Metrics for Each Exon Within Each Gene

The script also calculates metrics for individual exons within each gene. By breaking down the analysis to the exon level, the script provides a finer level of detail that can be useful for studying specific exonic regions, especially in clinical or functional genomics applications.

```
        # Initialize a dictionary to hold exon metrics for the gene
        exon_data_for_gene = {}

        for exon_name in gene_bed_df['EXON'].unique():
            # Initialize metrics for the exon
            exon_metrics = initialize_metrics()
            exon_bed_df = gene_bed_df[gene_bed_df['EXON'] == exon_name]
            exon_depths = pd.Series(dtype=float)

            exon_metrics['Size Coding'] = exon_bed_df['SIZE'].sum()

            for _, row in exon_bed_df.iterrows():
                start = row['START']
                end = row['END']
                exon_depths = pd.concat([
                    exon_depths,
                    depth_df[(depth_df['POS'] >= start) & (depth_df['POS'] <=
                    ↪  end)]['DEPTH']
                ], ignore_index=True)

            if len(exon_depths) > 0:
                # Calculate metrics for the exon
                exon_metrics['Average Read Depth'] = exon_depths.mean()
                exon_metrics['Min Read Depth'] = exon_depths.min()
                exon_metrics['Max Read Depth'] = exon_depths.max()
                exon_metrics['Size Covered'] = exon_depths.count()
                exon_metrics['Breadth of Coverage %'] = (
```

```
                            exon_metrics['Size Covered'] / exon_metrics['Size Coding']
                        ) * 100
                    else:
                        # Set metrics to zero if no depth data is available
                        exon_metrics['Average Read Depth'] = 0
                        exon_metrics['Min Read Depth'] = 0
                        exon_metrics['Max Read Depth'] = 0
                        exon_metrics['Size Covered'] = 0

                    # Calculate coverage percentages and intervals for the exon
                    coverage_counts_exon = {
                        threshold: (exon_depths >= threshold).sum() for threshold in
                        ↪  coverage_thresholds
                    }

                    for threshold in coverage_thresholds:
                        exon_metrics[f"Depth of Coverage % ({threshold}x)"] = (
                            coverage_counts_exon[threshold] / exon_metrics['Size Covered']
                        ) * 100 if exon_metrics['Size Covered'] > 0 else 0

                    depth_intervals_exon = {
                        "Depth of Coverage (>500x)": (exon_depths > 500).sum(),
                        # ... (other intervals)
                    }

                    exon_metrics.update(depth_intervals_exon)

                    # Store the exon metrics under the gene
                    exon_data_for_gene[exon_name] = exon_metrics

                # Store all exon metrics for the gene
                exons_data[gene] = exon_data_for_gene
```

Code 21: Calculating metrics for each exon within each gene.

This level of analysis is especially useful for pinpointing specific exons that may have insufficient coverage for accurate variant detection or expression analysis, which is critical in clinical settings where exonic variants can be associated with disease.

Compiling Final Results

Finally, the script compiles all the calculated metrics into a comprehensive results structure. This structure contains metrics for the entire set of genes combined (only if the analyses is Gene Panel), as well as per-gene and per-exon metrics. These results can then be used for visualization, reporting, or further analysis.

```
            # Store the overall metrics, gene metrics, and exon metrics in the results
            results[file_name]['All Genes'] = all_genes_metrics
            results[file_name]['Genes'] = genes_data
            results[file_name]['Exons'] = exons_data
```

```
5
6       return results
```

Code 22: Storing and returning the calculated metrics.

Error Handling and Edge Cases

Throughout the script, careful attention is given to error handling and edge cases to ensure robustness. For example, the script checks for missing or incomplete data in the session state, handles division by zero errors when calculating percentages, and sets zero as default value in the Dataframe when no Depth of Coverage data is available for a particular gene or exon.

2.3.6 Results Generation and Display Functionality

This section explains the `results.py` script, which is responsible for generating and displaying sequencing metrics using Streamlit. The script processes metrics data, organizes the results into tables, generates a plot, provides functionality to download a PDF report, and dynamically handles multiple samples. It allows users to interactively select and filter metrics, enhancing the flexibility and user-friendliness of the genomic data analysis workflow.

Importing Necessary Libraries and Displaying Logos

The script begins by importing the necessary libraries and modules required for its operation (Code 23). It imports standard libraries such as `pandas` for data manipulation, `numpy` for numerical operations, and `datetime` for handling date and time. It also imports custom components from the `components`, which includes `metrics`, `plot`, and `session_state` modules. Additionally, it imports `io` for inputoutput operations and `weasyprint` for generating PDF reports. The `base64` module is used for encoding images for embedding in the PDF.

```
1  import streamlit as st
2  import pandas as pd
3  from components import metrics, plot, session_state
4  import numpy as np
5  import io
6  from weasyprint import HTML
7  import base64
8  import datetime
```

Code 23: Importing necessary libraries and modules for the script.

Next, the script defines the file paths for the sidebar and main body logos (Code 24). These images are used to enhance the visual appeal of the application. The `st.logo()` function is utilized to display the logos within the Streamlit application. The `sidebar_logo` is displayed in the sidebar, while the `main_body_logo` serves as the icon image.

```
1  sidebar_logo = "data/img/unilabs_logo.png"
2  main_body_logo = "data/img/thumbnail_image001.png"
3  st.logo(sidebar_logo, icon_image=main_body_logo)
```

```
4
5 st.title("Results")
```

Code 24: Defining logo file paths and displaying them in the application.

Defining Metric Filters and Helper Functions

The script defines helper functions to facilitate the interactive selection and filtering of metrics displayed to the user. The `render_metric_filters` function is responsible for rendering the metric filters within a popover UI element (Code 25). It uses Streamlit's `st.popover` to create a popover window where users can select which metrics to display.

```
1  @st.fragment
2  def render_metric_filters(tab_name, metrics_dict):
3      with st.popover("Filters"):
4          st.subheader("Select Metrics to Display")
5
6          # Verifies if all metrics are selected
7          all_selected = all(
8              st.session_state.get(f"{tab_name}_metric_{metric}", False)
9              for metrics_list in metrics_dict.values()
10             for metric in metrics_list
11         )
12
13         # Button to select/deselect all metrics
14         if st.button("Select All" if not all_selected else "Deselect All",
           ↪  key=f"{tab_name}_select_all"):
15             select_all_metrics(not all_selected, metrics_dict, tab_name)
16             st.rerun()
17
18         # Divide the UI into 3 columns
19         col1, col2, col3 = st.columns(3)
20         metrics_keys = list(metrics_dict.keys())
21
22         # Display the metrics within the columns
23         for i, section in enumerate(metrics_keys):
24             with [col1, col2, col3][i % 3]:
25                 st.write(f"**{section}**")
26                 for metric in metrics_dict[section]:
27                     st.checkbox(metric, key=f"{tab_name}_metric_{metric}")
```

Code 25: Defining the 'render_metric_filters' function for metric selection.

This function first checks if all metrics are selected by iterating over the metrics in `metrics_dict` and checking their corresponding values in `st.session_state`. It then provides a button to select or deselect all metrics. The metrics are displayed in three columns for better readability, and each metric is represented as a checkbox.

Another helper function, `select_all_metrics`, is defined to handle the selection or deselection of all metrics (Code 26). This function updates the `st.session_state` for each

metric to reflect the user's choice.

```
def select_all_metrics(select_all, metrics_dict, key_prefix):
    """Function to select or deselect all metrics."""
    for section, metrics_list in metrics_dict.items():
        for metric in metrics_list:
            st.session_state[f"{key_prefix}_metric_{metric}"] = select_all
```

Code 26: Defining the 'select_all_metrics' function to toggle metric selection.

Defining the Desired Metrics Order

To ensure consistency in the presentation of metrics across different DataFrames, the script defines a list `desired_order` that specifies the order in which metrics should appear (Code 27).

```
# Desired order of metrics
desired_order = [
    'Size Coding', 'Size Covered', 'Breadth of Coverage %', 'Average Read Depth', 'Min Read Depth', 'Max Read Depth',
    'Depth of Coverage (0-1x)', 'Depth of Coverage (2-10x)', 'Depth of Coverage (11-15x)', 'Depth of Coverage (16-20x)',
    'Depth of Coverage (21-30x)', 'Depth of Coverage (31-50x)', 'Depth of Coverage (51-100x)',
    'Depth of Coverage (101-500x)', 'Depth of Coverage (>500x)',
    'Depth of Coverage % (1x)', 'Depth of Coverage % (10x)', 'Depth of Coverage % (15x)', 'Depth of Coverage % (20x)',
    'Depth of Coverage % (30x)', 'Depth of Coverage % (50x)', 'Depth of Coverage % (100x)', 'Depth of Coverage % (500x)'
]
```

Code 27: Defining the desired metrics order for display.

This ordered list was used later in the script when constructing DataFrames to display metrics in the specified sequence.

Calculating Metrics and Preparing DataFrames

The script calls the `calculate_metrics()` function from the `metrics` module to compute sequencing coverage metrics for one or multiple samples (Code 28). The results are stored in a dictionary, where each key corresponds to a sample file name, and the value is another dictionary containing metrics for that sample.

```
# Call the calculate_metrics function and store results for multiple depth files
results = metrics.calculate_metrics()
file_names = list(results.keys())
```

Code 28: Calculating metrics and obtaining results for samples.

To display metrics for all combined genes, the script constructs the DataFrame `all_genes_df` (Code 29). It starts by creating a list `all_metrics` that aligns with the

desired order of metrics. Then, the script initializes the DataFrame with a 'Metric' column populated by the metrics from `all_metrics`. For each sample, it retrieves the respective metrics from the `results` dictionary and adds them as new columns in the DataFrame, associating each sample's metrics to the corresponding file key.

```
# Prepare All Genes DataFrame
all_metrics = desired_order
all_genes_df = pd.DataFrame({'Metric': all_metrics})

for file_key in results:
    all_genes_metrics = results[file_key].get('All Genes', {})
    metrics_values = [all_genes_metrics.get(metric, np.nan) for metric in all_metrics]
    all_genes_df[file_key] = metrics_values
```

Code 29: Preparing the DataFrame for all genes metrics.

For gene-level metrics, the script compiles a set of all genes present in the results and creates individual DataFrames for each gene (Code 30). It iterates over the list of genes, initializes a DataFrame for each, and populates it with metrics from each sample.

```
# Prepare Genes DataFrames
all_genes_set = set()
for file_key in results:
    genes = results[file_key].get('Genes', {}).keys()
    all_genes_set.update(genes)
genes_list = sorted(all_genes_set)

genes_dfs = {}
for gene in genes_list:
    gene_metrics_df = pd.DataFrame({'Metric': desired_order})
    for file_key in results:
        gene_metrics = results[file_key].get('Genes', {}).get(gene, {})
        metrics_values = [gene_metrics.get(metric, np.nan) for metric in desired_order]
        gene_metrics_df[file_key] = metrics_values
    genes_dfs[gene] = gene_metrics_df
```

Code 30: Preparing individual DataFrames for each gene.

Similarly, for exon-level metrics, the script prepares DataFrames for each exon within each gene (Code 31). It constructs a nested dictionary `exons_dfs`, where the keys are gene names, and the values are dictionaries of exon DataFrames.

```
# Prepare Exons DataFrames
exons_dfs = {}
for gene in genes_list:
    exons_dfs[gene] = {}
    exons_set = set()
    for file_key in results:
        exons = results[file_key].get('Exons', {}).get(gene, {}).keys()
        exons_set.update(exons)
```

```
    exons_list = sorted(exons_set)

    for exon in exons_list:
        exon_metrics_df = pd.DataFrame({'Metric': desired_order})
        for file_key in results:
            exon_metrics = results[file_key].get('Exons', {}).get(gene, {}).get(exon, {})
            metrics_values = [exon_metrics.get(metric, np.nan) for metric in desired_order]
            exon_metrics_df[file_key] = metrics_values
        exons_dfs[gene][exon] = exon_metrics_df
```

Code 31: Preparing DataFrames for exon-level metrics.

These DataFrames are used later to display metrics in the application and generate reports.

Generating and Downloading the PDF Report

The script includes functionality to generate a PDF report of the metrics for a selected sample. It uses WeasyPrint to convert HTML content into a PDF. The report includes the Unilabs logo, the report generation date, and the metrics tables for the selected sample (Code 32).

```
# Download Report section
with st.status("Building the report...")as status:
    if len(file_names) > 1:
        # If there are multiple samples, allow the user to select one
        selected_sample = st.selectbox("Select Sample", file_names)
    else:
        # If there's only one sample, select it by default
        selected_sample = file_names[0]

        st.write(f'Download {selected_sample} report file (.pdf)')
    # Generate PDF report for the selected sample
    # Get the current date and time
    report_date = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    # Build an HTML string
    html_content = """
    <html>
    <head>
    <style>
    body {
        font-family: Arial, sans-serif;
    }
    table {
        border-collapse: collapse;
        width: 100%;
        font-size: 10pt;
    }
    th, td {
        text-align: left;
```

```
        padding: 8px;
        border: 1px solid #dddddd;
    }
    tr:nth-child(even) {
        background-color: #f9f9f9;
    }
    h2, h3 {
        color: #2F5496;
    }
    </style>
    </head>
    <body>
    """
    # Add Unilabs logo
    # Encode the logo image to base64
    with open(sidebar_logo, "rb") as image_file:
        encoded_string = base64.b64encode(image_file.read()).decode()
    html_content += f'<img src="{encoded_string}" alt="Unilabs Logo"
    ↪  style="width:200px;"><br><br>'
    # Add report generation date
    html_content += f'<p>Report generated on: {report_date}</p>'
    # Add Overview DataFrame for the selected sample
    html_content += f'<h2>Overview - Sample: {selected_sample}</h2>'
    if st.session_state.analysis in ['Gene Panel']:
        html_content += f'<p>Gene Panel: {st.session_state.panel_name}</p>'
        overview_df = all_genes_df[['Metric', selected_sample]]
        html_content += overview_df.to_html(index=False)
    if st.session_state.analysis in ['Exome']:
        html_content += f'<p>Exome</p>'
        overview_df = all_genes_df[['Metric', selected_sample]]
        html_content += overview_df.to_html(index=False)
    elif st.session_state.analysis in ['Single Gene']:
        # Add Gene DataFrames for the selected sample
        for gene_name, gene_df in genes_dfs.items():
            gene_sample_df = gene_df[['Metric', selected_sample]]
            if gene_sample_df[selected_sample].notna().any():
                html_content += f'<h2>Gene: {gene_name}</h2>'
                html_content += gene_sample_df.to_html(index=False)
        # Add Exon DataFrames for the selected sample
        for gene_name, exons_dict in exons_dfs.items():
            for exon_name, exon_df in exons_dict.items():
                exon_sample_df = exon_df[['Metric', selected_sample]]
                if exon_sample_df[selected_sample].notna().any():
                    html_content += f'<h3>Exon: {exon_name} (Gene: {gene_name})</h3>'
                    html_content += exon_sample_df.to_html(index=False)
    html_content += '</body></html>'
    # Convert HTML to PDF using WeasyPrint
    html_obj = HTML(string=html_content)
    pdf_bytes = html_obj.write_pdf()
    status.update(label="Generating PDF...")
```

```
    # Provide download button
    st.download_button(
        label="Download",
        data=pdf_bytes,
        file_name=f'report_{selected_sample}.pdf',
        mime='application/pdf'
    )
    status.update(label="Report ready for download", expanded=True)
```

Code 32: Generating and downloading a PDF report for the selected sample.

In this code, the script checks the number of samples and allows the user to select one if multiple samples are present. It then builds an HTML string that includes styling, the logo, the report date, and the metrics tables. The HTML content is converted to a PDF, and a download button is provided for the user to save the report.

Creating Tabs Based on Analysis Type

The script determines which tabs to display based on the type of analysis stored in `st.session_state.analysis` (Code 33). It supports analysis types such as "Gene Panel", "Single Gene", and "Exome". Depending on the analysis type, it creates appropriate tabs like "Overview", "Gene Detail", and "Exon Detail".

```
# Determine which tabs to display based on st.session_state.analysis
if st.session_state.analysis == 'Gene Panel':
    tab_names = ["Overview", "Gene Detail", "Exon Detail"]
elif st.session_state.analysis in ['Single Gene', 'Exome']:
    tab_names = ["Gene Detail", "Exon Detail"]
else:
    st.warning("Unsupported analysis type.")
    tab_names = []

# Create the tabs
tabs = st.tabs(tab_names)

# Map tab names to tab objects for easy reference
tab_dict = dict(zip(tab_names, tabs))
```

Code 33: Creating tabs based on the type of analysis.

This dynamic tab creation ensures that the application displays relevant information based on the user's analysis context.

Displaying Metrics in the "Overview" Tab

In the "Overview" tab, the script displays general information such as the report date, the analyzed files, and the gene panel name (Code 34). It uses the `render_metric_filters` function to allow users to select which metrics to display and the DataFrame is then displayed using `st.dataframe`.

```
if "Overview" in tab_dict:
    with tab_dict["Overview"]:
        st.write(f"Date: {report_date}")
        st.write(f"Analyzing file(s): {file_names}")
        st.write(f"Gene Panel: {st.session_state.panel_name}")

        if st.session_state.analysis in ['Gene Panel', 'Exome']:

            with st.container():
                # Use 'metrics_dict' instead of 'columns' to represent the metrics
                metrics_dict = {
                    "Basic Information": ["Average Read Depth", "Size Coding", "Size Covered",'Breadth of Coverage %', 'Min Read Depth','Max Read Depth'],
                    "Depth of Coverage": ["Depth of Coverage (0-1x)", "Depth of Coverage (2-10x)", "Depth of Coverage (11-15x)", "Depth of Coverage (16-20x)", "Depth of Coverage (21-30x)", "Depth of Coverage (31-50x)", "Depth of Coverage (51-100x)", "Depth of Coverage (101-500x)", 'Depth of Coverage (>500x)'],
                    "Depth of Coverage Percentage": ["Depth of Coverage % (1x)", "Depth of Coverage % (10x)", "Depth of Coverage % (15x)", "Depth of Coverage % (20x)","Depth of Coverage % (30x)", "Depth of Coverage % (50x)", "Depth of Coverage % (100x)", "Depth of Coverage % (500x)"]
                }

                # Initialize checkboxes as selected by default if not in session state
                for category in metrics_dict:
                    for metric in metrics_dict[category]:
                        if f"tab1_metric_{metric}" not in st.session_state:
                            st.session_state[f"tab1_metric_{metric}"] = True

                render_metric_filters("tab1", metrics_dict)

                # Selecting checked metrics
                selected_metrics = [
                    metric
                    for metrics_list in metrics_dict.values()
                    for metric in metrics_list
                    if st.session_state.get(f"tab1_metric_{metric}", False)
                ]
                final_metrics = ['Metric'] + [col for col in all_genes_df.columns if col != 'Metric']
                metrics_df = all_genes_df[all_genes_df['Metric'].isin(selected_metrics)].reset_index(drop=True)

                # Display the DataFrame
                st.dataframe(metrics_df[final_metrics], hide_index=True, height=842, width=800)
                if st.session_state.analysis in ['Gene Panel', 'Exome']:
                    if len(st.session_state.region) < 3:
```

```
                        plot.display_graphs()
                    else:
                        st.info("The plot was not generated due to the large volume of
                        ↪  data")
```

Code 34: Displaying metrics in the "Overview" tab with filters.

In this code, the script initializes metric checkboxes in the session state if they are not already set. It then renders the metric filters and displays the DataFrame containing the selected metrics. It also handles plotting, displaying a message if the data volume is too large.

Displaying Metrics in the "Gene Detail" Tab

In the "Gene Detail" tab, users can select a gene from a dropdown menu and view detailed metrics for that gene (Code 35). The script uses the same metric filtering mechanism as in the "Overview" tab.

```
if "Gene Detail" in tab_dict:
    with tab_dict["Gene Detail"]:
        st.write(f"Date: {report_date}")
        st.write(f"Analyzing file(s): {file_names}")

        with st.container():
            if not genes_list:
                st.warning("No genes found in the results.")
            else:
                gene = st.selectbox("Select Gene", genes_list, key="gene_selectbox")
                gene_metrics_df = genes_dfs[gene]

                # Filters for tab2
                metrics_dict = {
                    "Basic Information": ["Average Read Depth", "Size Coding", "Size
                    ↪  Covered",'Breadth of Coverage %', 'Min Read Depth','Max Read
                    ↪  Depth'],
                    "Depth of Coverage": ["Depth of Coverage (0-1x)", "Depth of Coverage
                    ↪  (2-10x)", "Depth of Coverage (11-15x)", "Depth of Coverage
                    ↪  (16-20x)","Depth of Coverage (21-30x)", "Depth of Coverage
                    ↪  (31-50x)", "Depth of Coverage (51-100x)", "Depth of Coverage
                    ↪  (101-500x)", 'Depth of Coverage (>500x)'],
                    "Depth of Coverage Percentage": ["Depth of Coverage % (1x)", "Depth of
                    ↪  Coverage % (10x)", "Depth of Coverage % (15x)", "Depth of Coverage %
                    ↪  (20x)","Depth of Coverage % (30x)", "Depth of Coverage % (50x)",
                    ↪  "Depth of Coverage % (100x)", "Depth of Coverage % (500x)"]
                }

                for category in metrics_dict:
                    for metric in metrics_dict[category]:
                        if f"tab2_metric_{metric}" not in st.session_state:
                            st.session_state[f"tab2_metric_{metric}"] = True

```

```
                render_metric_filters("tab2", metrics_dict)

                # Filter the DataFrame based on selected metrics
                selected_metrics = [
                    metric
                    for metrics_list in metrics_dict.values()
                    for metric in metrics_list
                    if st.session_state.get(f"tab2_metric_{metric}", False)
                ]
                df = gene_metrics_df[gene_metrics_df['Metric'].isin(selected_metrics)].res
                ↪  et_index(drop=True)
                final_metrics = ['Metric'] + [col for col in df.columns if col != 'Metric']

                # Display the DataFrame
                st.dataframe(df[final_metrics], hide_index=True, height=842, width=800)
                if st.session_state.analysis in ['Single Gene']:
                    plot.display_graphs()
```

Code 35: Displaying metrics in the "Gene Detail" tab with filters.

Displaying Metrics in the "Exon Detail" Tab

In the "Exon Detail" tab, users can select a gene and then an exon to view detailed metrics for that exon (Code 36). The script ensures that exons are associated with the selected gene and uses the metric filtering mechanism.

```
if "Exon Detail" in tab_dict:
    with tab_dict["Exon Detail"]:
        st.write(f"Date: {report_date}")
        st.write(f"Analyzing file(s): {file_names}")

        with st.container():
            if not genes_list:
                st.warning("No genes found in the results.")
            else:
                gene = st.selectbox("Select Gene", genes_list, key="exon_gene_selectbox")
                exons_list = sorted(exons_dfs[gene].keys())
                if not exons_list:
                    st.warning(f"No exons found for gene {gene}.")
                else:
                    exon = st.selectbox("Select Exon", exons_list, key="exon_selectbox")
                    exon_metrics_df = exons_dfs[gene][exon]

                    # Filters for tab3
                    metrics_dict = {
                        "Basic Information": ["Average Read Depth", "Size Coding", "Size
                        ↪  Covered",'Breadth of Coverage %', 'Min Read Depth','Max Read
                        ↪  Depth'],


```

```
                    "Depth of Coverage": ["Depth of Coverage (0-1x)", "Depth of
                    ↪  Coverage (2-10x)", "Depth of Coverage (11-15x)", "Depth of
                    ↪  Coverage (16-20x)","Depth of Coverage (21-30x)", "Depth of
                    ↪  Coverage (31-50x)", "Depth of Coverage (51-100x)", "Depth of
                    ↪  Coverage (101-500x)", 'Depth of Coverage (>500x)'],
                    "Depth of Coverage Percentage": ["Depth of Coverage % (1x)", "Depth
                    ↪  of Coverage % (10x)", "Depth of Coverage % (15x)", "Depth of
                    ↪  Coverage % (20x)","Depth of Coverage % (30x)", "Depth of
                    ↪  Coverage % (50x)", "Depth of Coverage % (100x)", "Depth of
                    ↪  Coverage % (500x)"]
                }

                for category in metrics_dict:
                    for metric in metrics_dict[category]:
                        if f"tab3_metric_{metric}" not in st.session_state:
                            st.session_state[f"tab3_metric_{metric}"] = True

                render_metric_filters("tab3", metrics_dict)

                # Filter the DataFrame based on selected metrics
                selected_metrics = [
                    metric
                    for metrics_list in metrics_dict.values()
                    for metric in metrics_list
                    if st.session_state.get(f"tab3_metric_{metric}", False)
                ]
                df = exon_metrics_df[exon_metrics_df['Metric'].isin(selected_metrics)]
                ↪  .reset_index(drop=True)
                final_metrics = ['Metric'] + [col for col in df.columns if col !=
                ↪  'Metric']

                # Display the DataFrame
                st.dataframe(df[final_metrics], hide_index=True, height=842, width=800)
```

Code 36: Displaying metrics in the "Exon Detail" tab with filters.

This code segment ensures that users can drill down to exon-level details, providing a comprehensive view of the sequencing metrics.

2.4 Deployment

The deployment of the software application was streamlined using Docker [53], a containerization platform that creates isolated, lightweight, and reproducible environments. Below is a detailed explanation of a `Dockerfile`, used to package the application and its dependencies in a self-contained environment. This section provides an overview of the Docker configuration and explains the steps involved in the deployment process.

2.4.1 Base Image and Working Directory

The deployment begins by selecting the slim version of Python 3.12.1 as the base image, which provides a minimal and optimized environment suited for Python applications. The `WORKDIR` directive sets the working directory to `/metrics_app`, ensuring that all subsequent operations during the Docker build process occur within this directory. This strategy helps avoid nested directories within the container and maintains a clean and straightforward project structure (Code 37).

```
FROM python:3.12.1-slim

# Set working directory
WORKDIR /metrics_app
```

Code 37: Dockerfile: Setting the base image and working directory.

2.4.2 Installing System Dependencies

To ensure the correct functionality of the application and its various components, essential system dependencies are installed using `apt-get`. This step installs development tools such as `build-essential`, along with libraries like `zlib1g-dev`, `libbz2-dev`, `libcurl4-openssl-dev`, and `libjpeg62-turbo-dev`, which are necessary for handling compressed files, secure connections, and image processing. Additional libraries like `libcairo2` and `libpango` are required by `weasyprint` for rendering PDFs. After installing these dependencies, cached files are removed using `rm -rf varlibaptlists*`, reducing the image size and improving the overall security (Code 38).

```
# Install system dependencies
RUN apt-get update && apt-get install -y \
    curl \
    build-essential \
    zlib1g-dev \
    libbz2-dev \
    liblzma-dev \
    libcurl4-openssl-dev \
    libssl-dev \
    libncurses5-dev \
    libncursesw5-dev \
    autoconf \
    automake \
    libcairo2 \
    libpango-1.0-0 \
    libpangocairo-1.0-0 \
    libpangoft2-1.0-0 \
    libffi-dev \
    libjpeg62-turbo-dev \
    libgdk-pixbuf2.0-0 \
    libgdk-pixbuf2.0-dev \
```

```
    shared-mime-info \
    && rm -rf /var/lib/apt/lists/*
```

Code 38: Dockerfile: Installing system dependencies.

2.4.3 Installing Miniconda and Python Environment Setup

Miniconda is installed within the container to manage Python packages and system dependencies more effectively. By running a script that downloads and installs Miniconda in a non-interactive mode, the base environment is set up under `/opt/conda`. The `conda` executable is then linked to the system's `/usr/local/bin` path to ensure easy access during subsequent operations. This setup provides a lightweight and flexible Python environment for managing dependencies via the `environment.yml` file (Code 39).

```
# Install Miniconda
RUN curl -L https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -o
↪  miniconda.sh && \
    bash miniconda.sh -b -p /opt/conda && \
    rm miniconda.sh && \
    /opt/conda/bin/conda clean --all && \
    ln -s /opt/conda/bin/conda /usr/local/bin/conda

# Add Conda to the PATH
ENV PATH=/opt/conda/bin:$PATH

# Copy project files into the container
COPY . .

# Create and activate the conda environment
RUN conda env create -f environment.yml && conda clean -a
```

Code 39: Dockerfile: Installing Miniconda and setting up the Python environment.

2.4.4 Installing Samtools

Samtools, a critical tool for genomic data manipulation, is installed from source within the Conda environment. The source code is downloaded and extracted from the official Samtools repository. The software is then compiled using `make`, and the binaries are installed into the Conda environment under `/opt/conda/envs/metrics_app_env`, ensuring that it is isolated from the system-wide environment and available for genomic analysis within the project (Code 40).

```
# Install Samtools inside the Conda environment
RUN /bin/bash -c "source activate metrics_app_env && \
    curl -L
    ↪  https://github.com/samtools/samtools/releases/download/1.19/samtools-1.19.tar.bz2 |
    ↪  tar -xj && \
    cd samtools-1.19 && \
```

```
    ./configure --prefix=/opt/conda/envs/metrics_app_env && \
    make && \
    make install && \
    cd .. && rm -rf samtools-1.19"
```

Code 40: Dockerfile: Installing Samtools.

2.4.5 Exposing the Application Port and Healthcheck

The `EXPOSE` directive is used to make port 8501 accessible, which is the default port for Streamlit applications. This ensures that the application will be available on this port once the container is running. Additionally, a health check mechanism is implemented using the `HEALTHCHECK` directive. The health check periodically sends requests to the Streamlit application's health endpoint to verify that the service is operational. If the service fails, the container will be marked as unhealthy, allowing for automated recovery mechanisms (Code 41).

```
# Expose the Streamlit port
EXPOSE 8501

# Healthcheck for Streamlit
HEALTHCHECK CMD curl --fail http://localhost:8501/_stcore/health || exit 1
```

Code 41: Dockerfile: Exposing application port and setting up health checks.

2.4.6 Entry Point and Application Execution

Finally, the `ENTRYPOINT` directive is used to define the command that runs when the container starts. The Conda environment is activated using the `conda run` command, followed by the execution of Streamlit. The Streamlit server is launched using the `Home.py` script located in the `app` folder, and it listens on port 8501 for incoming requests. The `-server.port=8501` and `-server.address=0.0.0.0` flags ensure that the application is accessible from any network interface, allowing external connections to the containerized application (Code 42).

```
# Set the entrypoint for Streamlit
ENTRYPOINT ["conda", "run", "--no-capture-output", "-n", "metrics_app_env", "streamlit",
↪  "run", "app/Home.py", "--server.port=8501", "--server.address=0.0.0.0"]
```

Code 42: Dockerfile: Setting entry point and application execution.

CHAPTER 3

Results

"It always seems impossible until it's done." - Nelson Mandela, former President of South Africa

3.1 Evolution of Software Development

The software development process has undergone several important stages, each enhancing its functionality and usability. The figures below illustrate the application's progress, showcasing different phases of refinement as the project matured.

3.1.1 Initial Stages: Basic Functionality and User Interaction

The first figure (Figure 3.1) represents the early stages of development, where the primary focus was on creating a user-friendly interface for calculating Average Read Depth and Depth of Coverage at diferent threshold from a BAM file for a Single Gene. In this version, the application prominently features a two-step process where the user selects a BED file releated to the gene to analyse and one or more BAM files. These files are then processed to calculate key metrics, which are displayed in a results table. This simple design allowed for efficient user interaction but was somewhat limited in terms of flexibility and scope.

At this stage, the core challenge was to ensure that the software could handle large genomic files while presenting the results in a clear and intuitive format. The layout emphasizes simplicity, making it easier for users to navigate through the two-step process. However, as the project progressed, the need for more advanced features became evident.

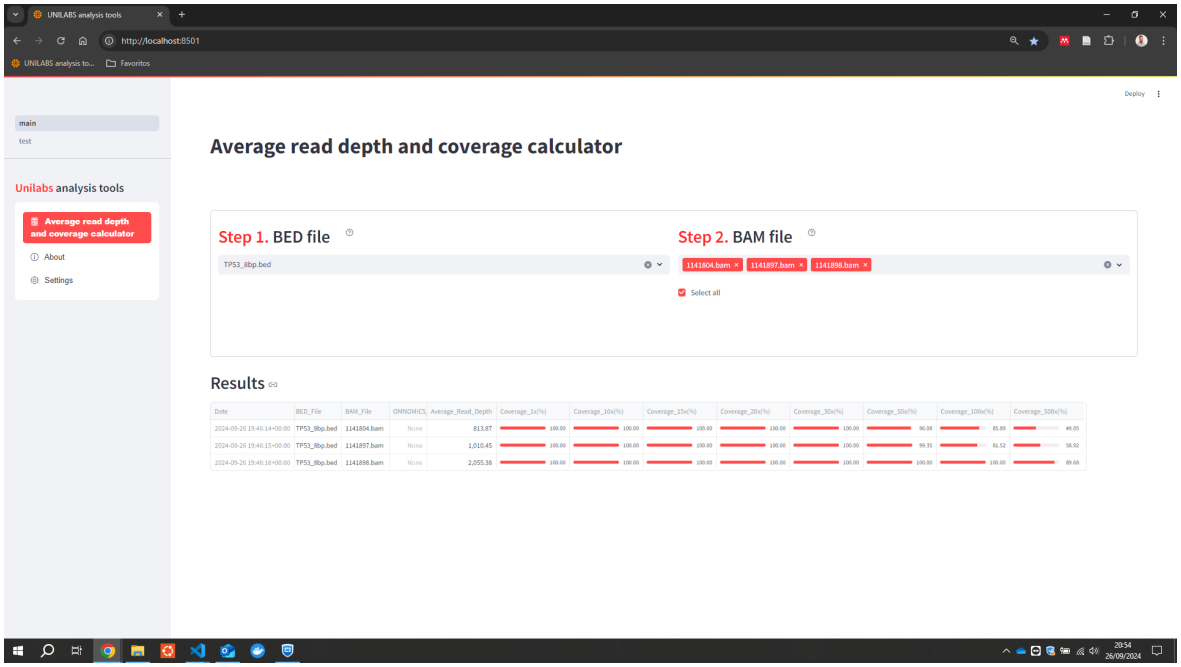


Figure 3.1: First version of the GUI.

3.1.2 Refinement: Introducing Flexibility and Multiple Analysis Modes

In the second figure (Figure 3.2), the software has significantly evolved to include more detailed analysis options. The interface now presents multiple analysis types: *Single Gene*, *Gene Panel*, and *Exome*, catering to different research needs. This flexibility represents a major shift from the earlier version, as it now allows users to select specific genome assemblies and regions of interest by using an Universal BED file. Additionally, the results section has been split into tabs such as *Overview* and *Exon Details*, giving users the ability to drill down into the metrics for a the gene or explore exon-level coverage details. However, even though this last features were thinked to be used in this version, they only have been work fully in the last version of the software.

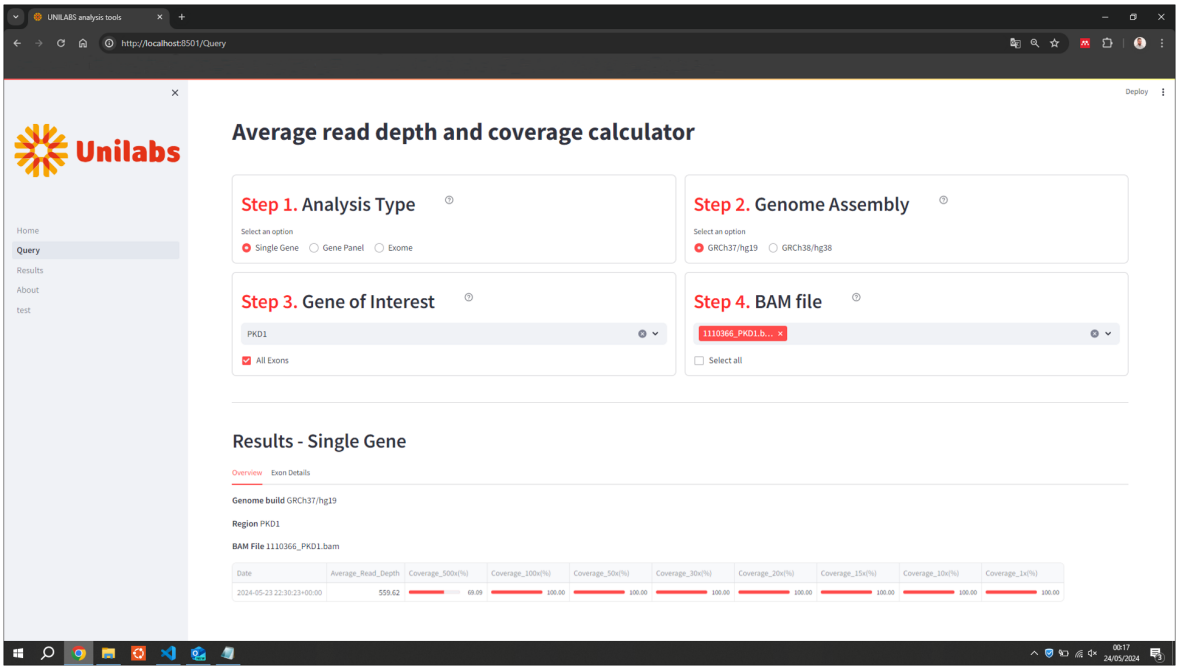


Figure 3.2: Second version of the GUI.

The final version of the software is presented in the next section in detail with real-world data to showcase its full capabilities and effectiveness in genomic analysis. This final iteration represents the culmination of numerous improvements in both the user interface and the backend computational logic.

3.1.3 Overview of the Final Version

The final version of the software maintains the core functionality established in earlier versions, such as calculating Average Read Depth and Depth of Coverage from BED and BAM files. However, it has evolved to include more refined features, enhanced performance, and the ability to handle more complex datasets.

This section presents a detailed overview of the final version of the software using real-world genomic data, illustrating its capabilities in calculating Depth and Breadth of Coverage for genomic regions of interest. The images provided demonstrate the final interface and processing stages.

Login Interface

As seen in Figure 3.3, the software begins with a user authentication process, offering a simple and clean login interface. This feature, although optional in the final deployment, ensures that access to sensitive data is controlled. However, it is important to note that the current implementation of the login interface is a prototypical version and does not offer any form of security. The system does not perform encryption or validation of credentials beyond basic checks, and therefore it should not be relied upon for securing sensitive information. Further development would be required to implement a secure authentication system if needed in future deployments.

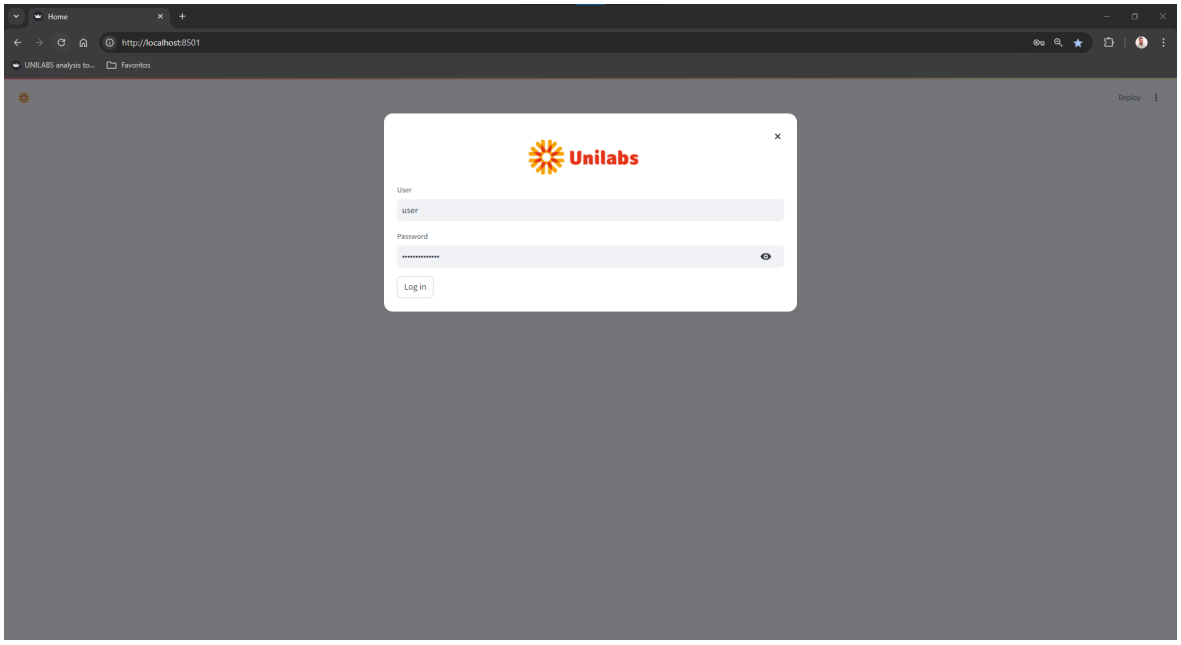


Figure 3.3: Login Interface for the Software

The next step of the process involves selecting the analysis type, as shown in Figure 3.4. Users can choose between three options: Single Gene, Gene Panel, and Exome. This selection determines the scope of the analysis and the subsequent steps in the workflow.

Single Gene Analysis

In this version, users can perform a single gene analysis by selecting the appropriate options for genome assembly and gene of interest. The software also provides the option to analyze all exons within the selected gene or focus on specific exons of interest. BAM/CRAM files containing the sequencing data are acessed and processed by Samtools for detailed metrics.

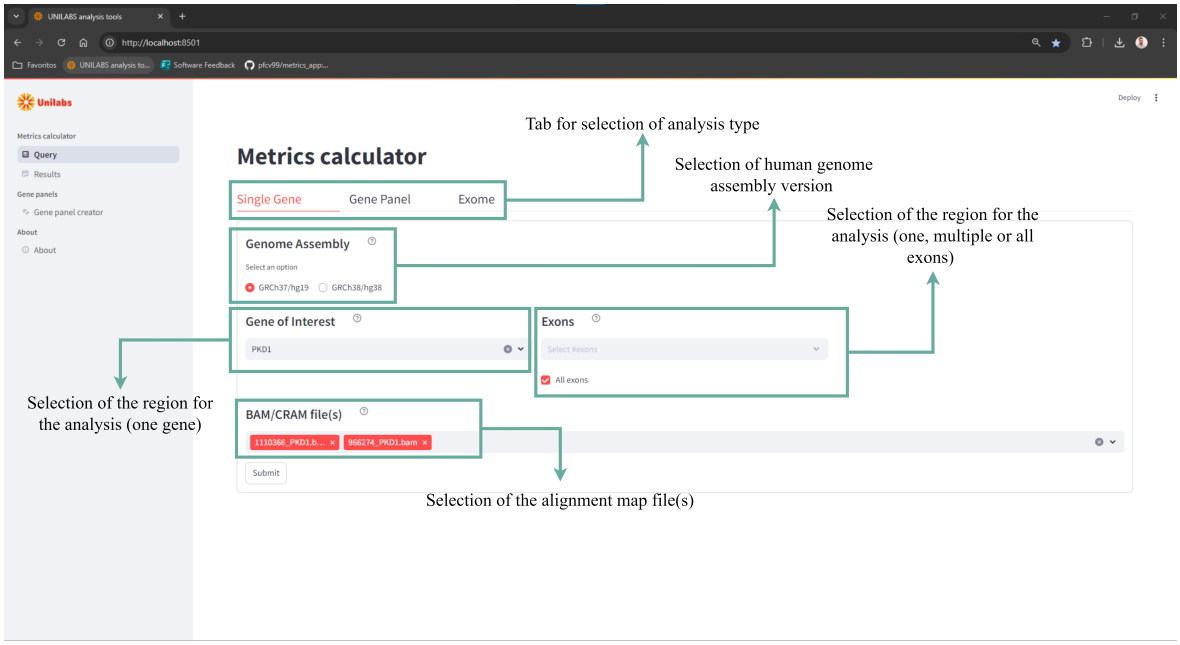


Figure 3.4: Single Gene Analysis Workflow

- **Results and Report Generation**

  Once the data is processed, users can access the results in the `Results` tab, as seen in Figure 3.5. The software compiles a detailed report that includes various metrics such as Average Read Depth, Breadth of Coverage, and Depth of Coverage percentages across different thresholds (e.g., 10x, 20x, 30x). This data is also available for download in a CSV or PDF format, ensuring users can retain a permanent copy of the analysis results.

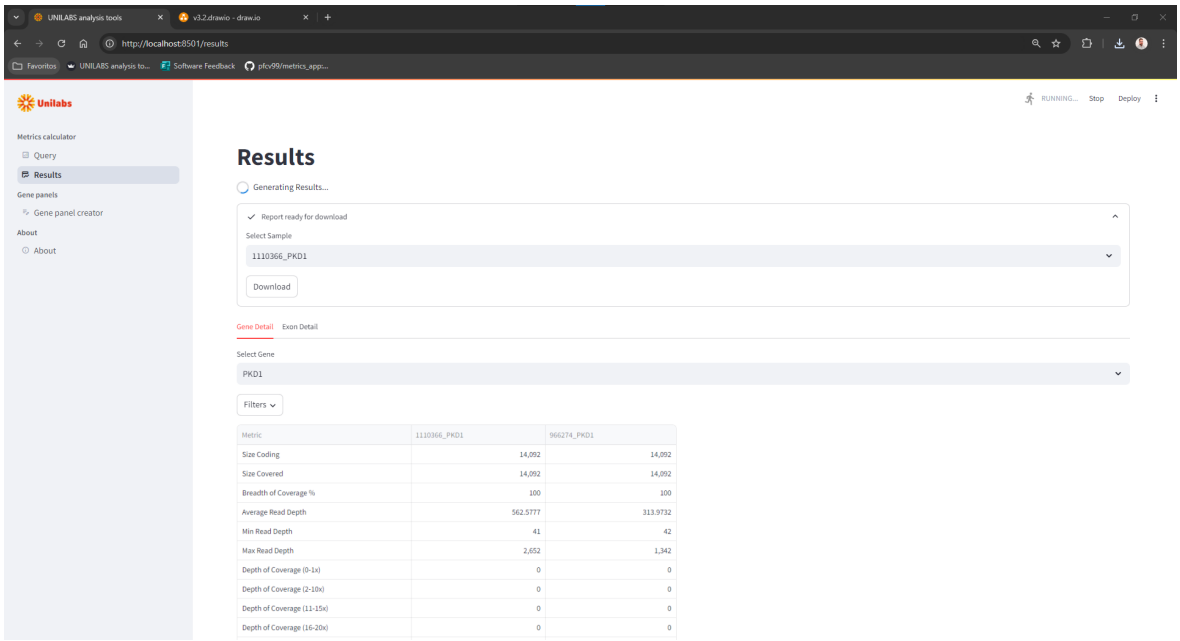


Figure 3.5: Results Tab Loading the Final Report

In Figure 3.5 and 3.6, the detailed metrics for the gene *PKD1* are displayed, of-

fering both gene-level and exon-level statistics. This comprehensive breakdown allows researchers to thoroughly assess the sequencing coverage for the analyzed samples. Key metrics include the Size Coding of the gene, minimum and Maximum Read Depth, and coverage percentages across various depth thresholds, providing valuable insights into the quality and completeness of the sequencing data.

For this case study, the Size Coding of the *PKD1* gene is 14092 Base Pair (BP), with a Size Covered of 14092 BP for both samples, resulting in a Breadth of Coverage of 100%. The Average Read Depth across the three samples was 562.58x and 313.98x, respectively. Depth of Coverage, or the number of times a particular region of the genome is covered by reads, directly impacts the reliability of variant detection, gene expression studies, and other genomic analyses. Inconsistent or insufficient depth may lead to variability in the ability to accurately detect mutations or copy number variations, potentially resulting in false positives or false negatives. For instance, lower depth may miss variants that are present at low frequencies, while higher depth ensures that even rare mutations are confidently identified. [21]

When examining the Depth of Coverage across different thresholds, the percentages for the 10x, 20x, and 30x thresholds were consistently 100% for both samples, demonstrating that all regions of interest achieved sufficient coverage at these lower thresholds. However, at higher thresholds, the Depth of Coverage showed more variability. For the 50x threshold, the coverage percentages were 99.99% and 99.47%, respectively. Similarly, at the 100x threshold, the Depth of Coverage percentages were 98.92% and 91.69%. Finally, at the highest threshold of 500x, the coverage percentages dropped more significantly, with values of 51.36% and 14.66%, respectively.

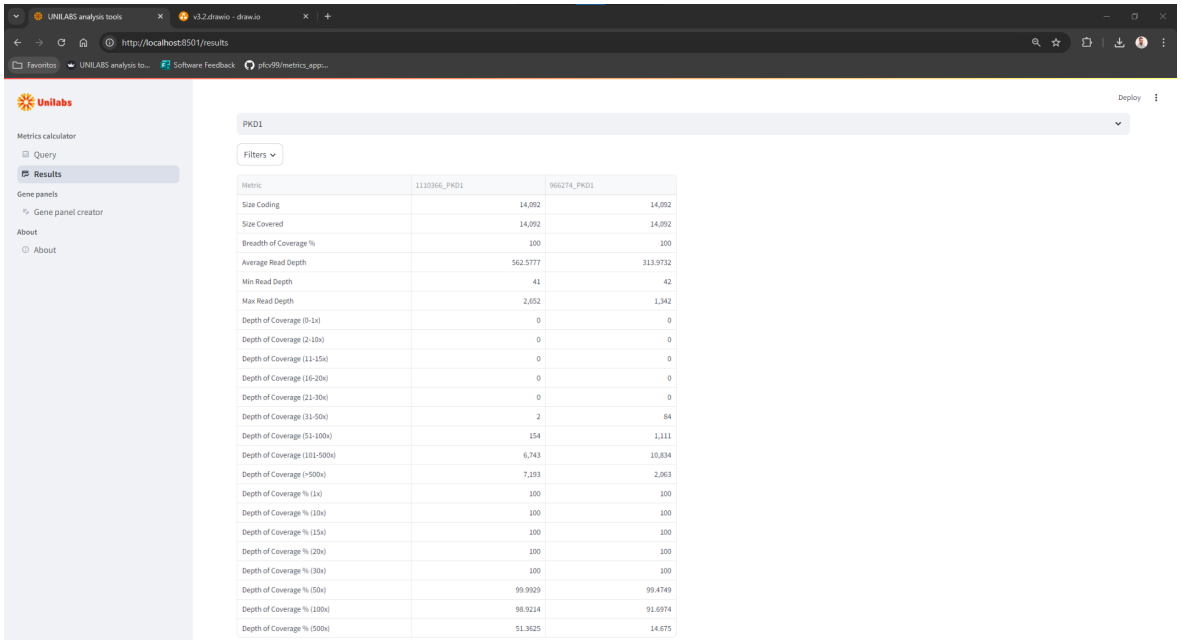


Figure 3.6: Final Report with Detailed Metrics for Gene TP53

- **Depth of Coverage Visualization**

One of the critical features of the final software version is its ability to visualize in a plot the distribution of Depth of Coverage across each position, as shown in Figure 3.7. This visualization allows users to see the depth of sequencing across the gene of interest, with blue regions highlighting exons. A threshold can be set by the user, and regions that fall below this threshold are highlighted in red, ensuring that gaps or low-coverage areas are easily identified. This is particularly important for researchers assessing the completeness of their sequencing data.

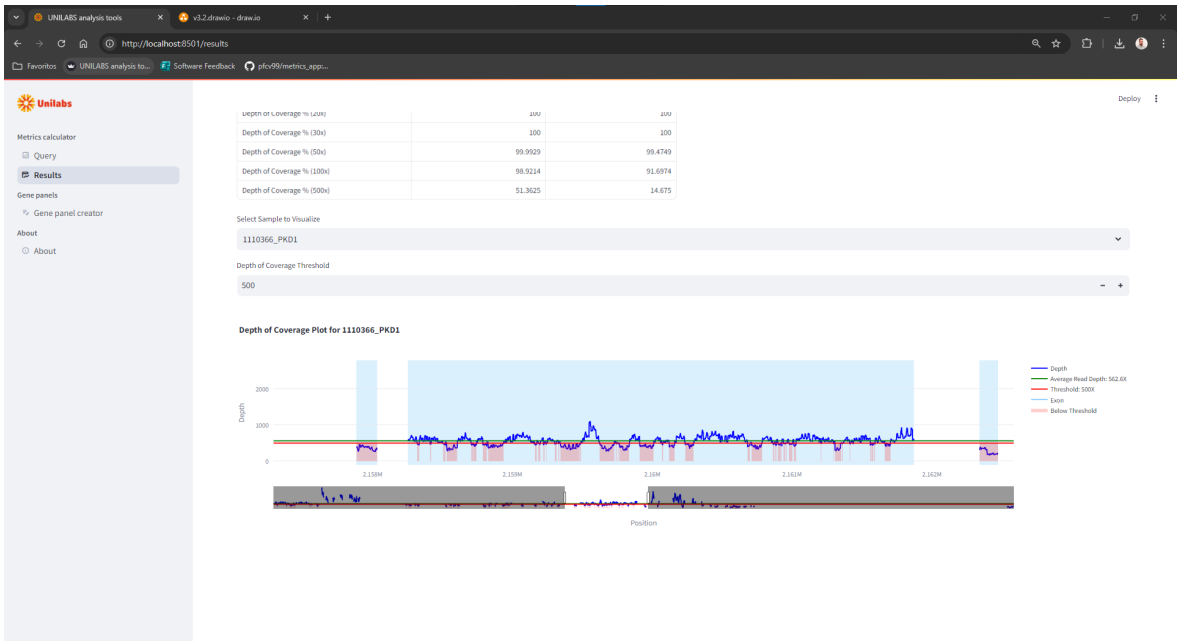


Figure 3.7: Depth of Coverage Visualization for Gene TP53

Gene Panel Analysis

In the gene panel analysis, the software is configured to process multiple genes simultaneously, as opposed to the single gene analysis. This functionality is particularly useful when studying gene panels associated with specific hereditary diseases, such as the BRCA1 and BRCA2 genes, commonly linked to breast and ovarian cancer. The analysis workflow is similar to that of a single gene but extends to multiple regions of interest, providing broader insights into the Depth and Breadth of Coverage across the entire panel.

- **Panel Selection and Input**

  The user begins by selecting the appropriate genome assembly and the gene panel of interest. For this case study, the panel for breast and ovarian cancer was selected, which includes 27 genes, among which the BRCA1 and BRCA2 genes. The input consists of a BAM or CRAM file associated with the selected panel, as shown in Figure 3.8.

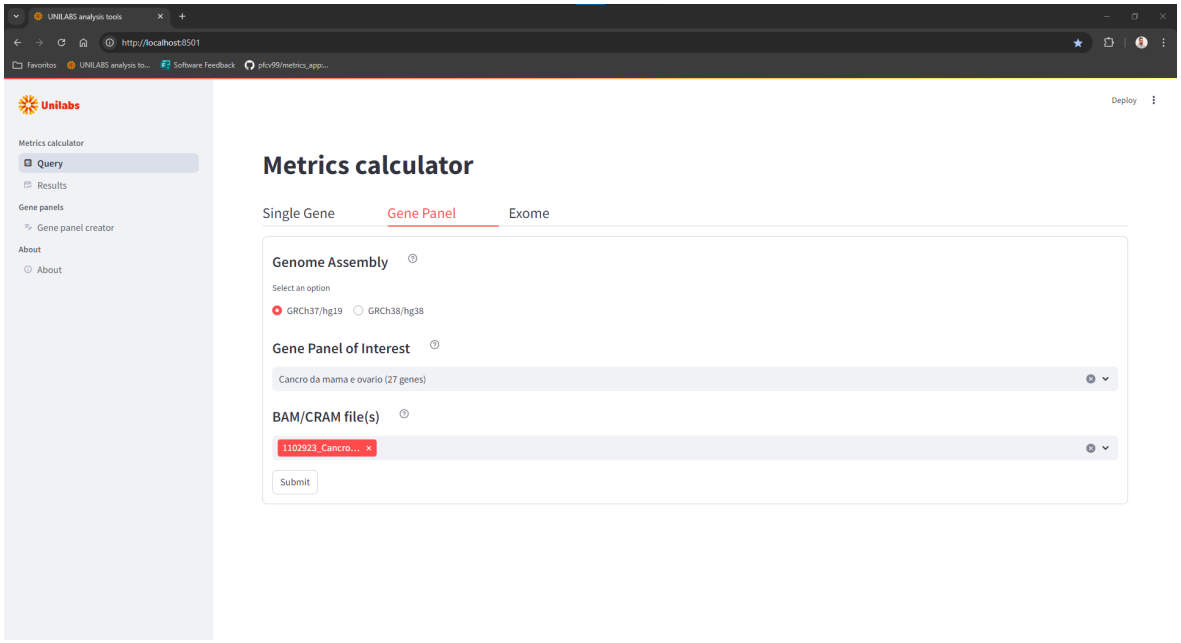


Figure 3.8: Gene Panel Input Selection and Submission

- **Overall Results for the Gene Panel**

  Once the data is processed, the software generates a comprehensive table of metrics for the entire gene panel. This includes critical metrics such as Size Coding, Size Covered, Breadth of Coverage, and Average Read Depth. These metrics are essential for evaluating the quality of sequencing across all genes in the panel. Figure 3.9 illustrates the results generated for the gene panel associated with hereditary breast and ovarian cancer. For this specific case, a panel with a Size Coding of 119175 BP, the Size Covered was 93043 BP, resulting in a Breadth of Coverage of 78.07%. The Average Read Depth across the panel was 263.93x, with a Minimum Read Depth of 1x and a Maximum Read Depth of 631x. The Depth of Coverage across different thresholds revealed that the panel achieved a Depth of Coverage between 87.28% and 100% at the 1x, 10x and 15x, 20x, 30x, 50x and 100x threshold. For the 500x threshold, the Depth of Coverage percentage dropped to 1.9%, indicating regions less covered by sequencing.

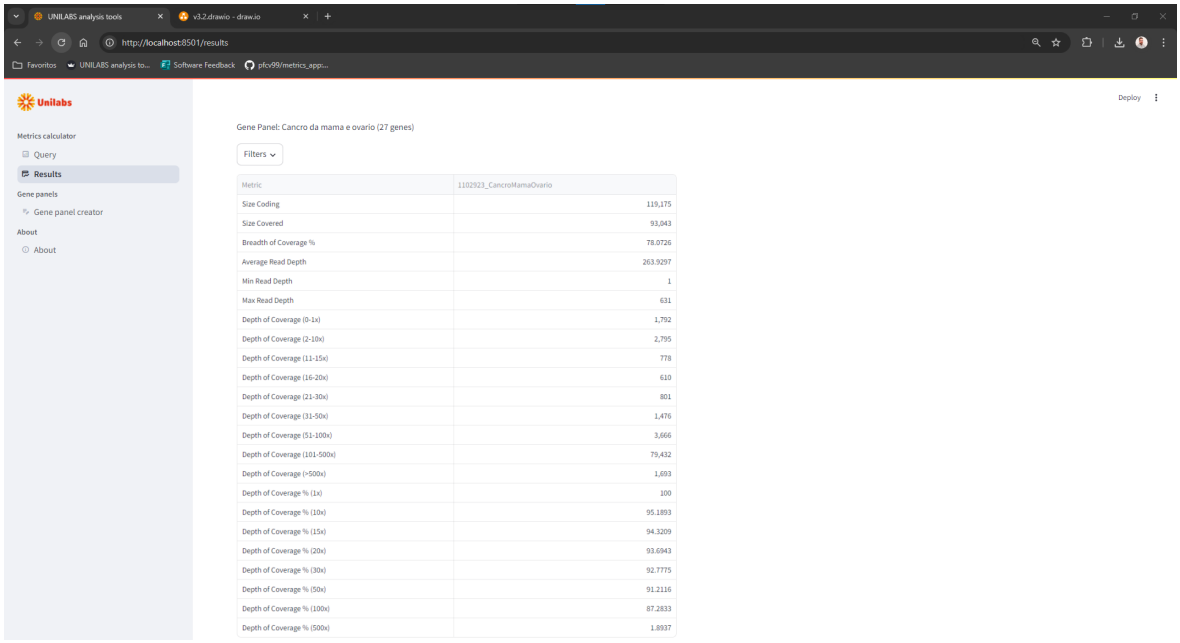


Figure 3.9: Overall Gene Panel Results for Hereditary Breast and Ovarian Cancer

- **Individual Gene Metrics**

  The software allows users to dive deeper into the metrics for individual genes within the panel. Figure 3.10 displays the results for the BRCA1 gene. The Size Coding of BRCA1 was 6343 BP, with a Size Covered of 6124 BP, resulting in a Breadth of Coverage of 96.5%. The Average Read Depth for BRCA1 was 345.3x, with a Minimum Read Depth of 1x and a Maximum Read Depth of 533x. The Depth of Coverage across different thresholds showed consistent percentages above 99% for the 1x, 10x,, 15x, 20x, 30x, 50x and 100x thresholds, with a slight drop to 1.6% at the 500x threshold.

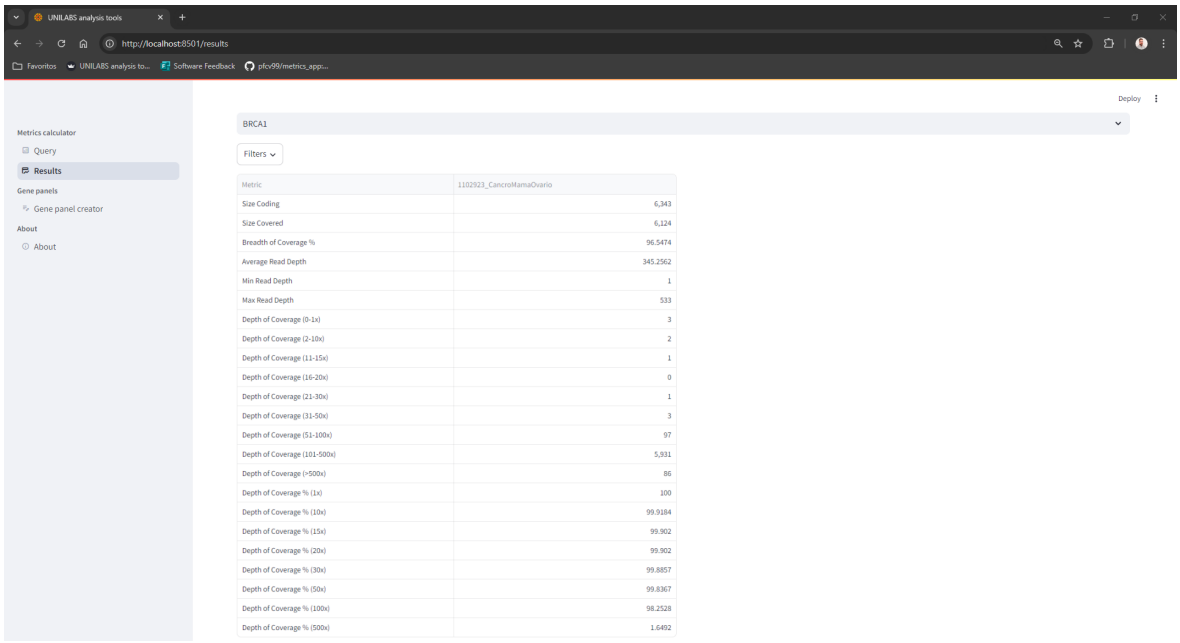


Figure 3.10: Detailed Metrics for the BRCA1 Gene

- **Exon-Level Analysis**

  In addition to gene-level metrics, the software also offers exon-level analysis. Users can select specific exons within the genes to obtain a more granular view of the sequencing Depth of Coverage. This level of detail is particularly useful when assessing the completeness of the sequencing across critical regions within each gene. Figure 3.11 shows the results for the 13rd exon of the BRCA1 gene, where key metrics are also displayed. This exon-level detail allows researchers to pinpoint regions that may require additional sequencing or validation. For the 13rd exon of BRCA1, the Size Coding was 126 BP, with a Size Covered of 126 BP, resulting in a Breadth of Coverage of 100%. The Average Read Depth for this exon was 261.98x, with a Minimum Read Depth of 216x and a Maximum Read Depth of 291x. The Depth of Coverage across different thresholds showed consistent coverage percentages of 100% for the 1x, 10x, 15x, 20x, 30x, 50x, 100x thresholds, with a drop to 0% at the 500x threshold.

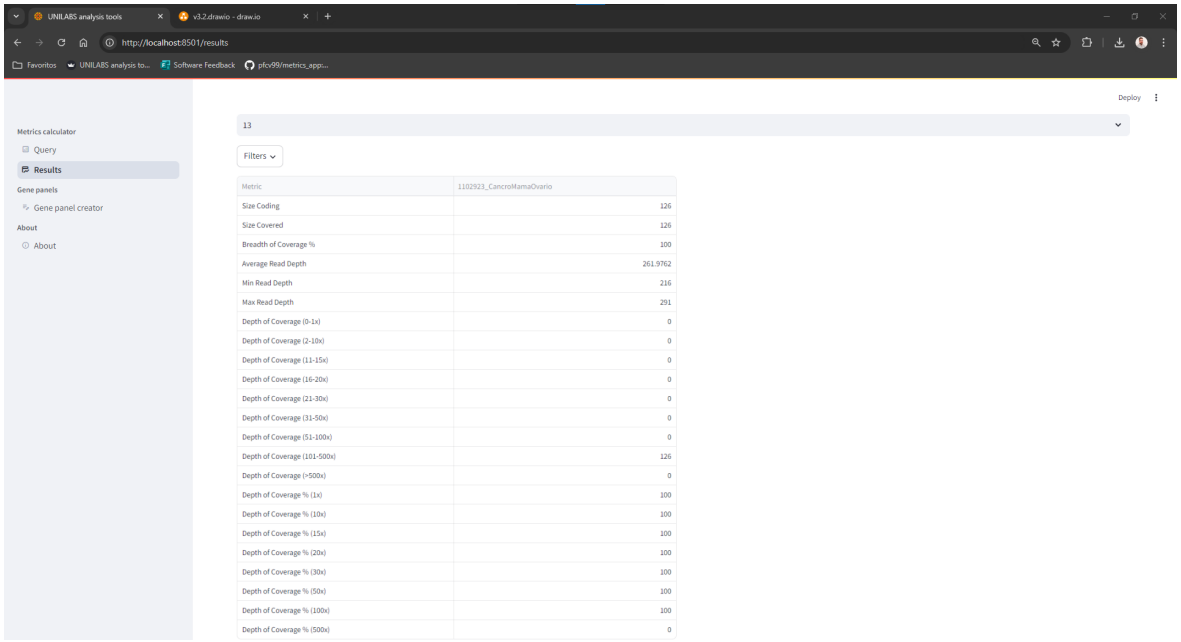


Figure 3.11: Exon-Level Metrics for the BRCA1 Gene

The gene panel analysis feature provides a powerful tool for evaluating sequencing coverage across multiple genes simultaneously. By offering both gene-level and exon-level insights, the software ensures that researchers can thoroughly assess the quality of their sequencing data, identify potential gaps in coverage, and make informed decisions regarding further analysis or re-sequencing.

3.2 Test and Validation

To validate the tool, each BAM file was analyzed using the commercial software Omnomics, and the results were compared with those obtained from the developed software. The analyses were performed separately for a Single Gene and a Gene Panel, ensuring that each BAM file was used for its respective analysis.

3.2.1 Single Gene Analysis

The gene TP53 was selected for the Single Gene analysis, and the results obtained from the developed software were compared with those from Omnomics. The metrics compared included Average Read Depth and Depth of Coverage % (1x, 10x, 20x, 30x, 50x, 100x, 500x).

Table 3.1: Comparison of Metrics between the developed software and Omnomics for Gene TP53

Metric	**Developed software**	**Omnomics**
Average Read Depth	874,97	891
Depth of Coverage % (1x)	99.86	100
Depth of Coverage % (10x)	99.86	100
Depth of Coverage % (20x)	99.86	100
Depth of Coverage % (30x)	99.86	100
Depth of Coverage % (50x)	99.86	100
Depth of Coverage % (100x)	99.79	99.90
Depth of Coverage % (500x)	57.42	59

The comparison of results between the two software solutions reveals some minor discrepancies. In the Average Read Depth metric, the developed software reported an average depth of 874.97, whereas Omnomics recorded a slightly higher value of 891. These differences can be primarily attributed to the variations in the reference BED files used by each tool. The developed software relies on an optimized BED file from MANE, while Omnomics utilizes a GENCODE BED created through the Table Browser tool of the Genome Browser, specifically targeting the TP53 gene with an extension of 8 base pairs. These differing BED files lead to variations in the covered regions, directly influencing metrics such as Average Read Depth.

Both tools exhibited identical results for the Depth of Coverage percentages across lower thresholds, including 1x, 10x, 20x, 30x, and 50x, demonstrating full coverage at these depths. However, slight deviations are noted at higher coverage thresholds. For instance, the Depth of Coverage percentage at 100x reveals minimal differences, with the developed software reporting 99.79% and Omnomics reporting 99.9%.

A more noticeable difference is observed in the Depth of Coverage percentage at 500x, where the developed software reported 57.42%, compared to Omnomics 59%. This discrepancy can once again be attributed to the differing regions encompassed by the respective BED files, which affect the depth distribution across the analyzed regions.

Overall, by comparing the developed tool with Omnomics, it was possible to infere that the developed software is capable of providing accurate and reliable metrics for Single Gene analysis. The variations observed in the results are primarily due to differences in the reference BED files used by each tool, highlighting the importance of selecting the appropriate reference file for accurate coverage analysis.

3.2.2 Gene Panel Analysis

For the analysis of a gene panel the previous panel "Cancro da mama e ovário (27 genes)" was used. One of the BAM files used was compared between the the developed software

and the commercial tool Omnomics. The results are detailed in Table 3.2, highlighting the differences between the two tools for several metrics.

Table 3.2: Comparison of Metrics between the developed software and Omnomics for Gene Panel: Cancro da mama e ovário (27 genes)

Metric	**Developed software**	**Omnomics**
Average Read Depth	263.93	388
Depth of Coverage % (1x)	100	99.60
Depth of Coverage % (10x)	95.19	99.50
Depth of Coverage % (20x)	93.69	99.40
Depth of Coverage % (30x)	92.78	99.20
Depth of Coverage % (50x)	91.21	98.80
Depth of Coverage % (100x)	87.28	96.10
Depth of Coverage % (500x)	1.89	27.60

The comparison between the developed software and Omnomics for this gene panel reveals several key differences in the reported metrics. Most notably, the Average Read Depth shows a significant disparity, with the developed software reporting a depth of 263.93, while Omnomics reports a substantially higher value of 388. This difference can once again be attributed to the different BED files used by each tool.

For the Depth of Coverage % metrics, both tools report similar values, though some minor differences are present. At the 1x coverage threshold, the developed software reports 100%, while Omnomics reports 99.6%. Similarly, for the 10x, 20x, 30x, and 50x thresholds, the differences remain small, with Omnomics consistently reporting slightly higher percentages. However, at the 100x threshold, the tools report nearly identical values, with the developed software at 87.28% and Omnomics at 96.1%.

The most significant divergence occurs at the 500x threshold. The developed software reports a very low value of 1.89%, while Omnomics reports a much higher 27.6%. This discrepancy is likely due to the differing regions targeted by the BED files, with Omnomics potentially including regions of higher coverage within the panel that are not present in the MANE BED file used by the developed software.

Overall, the comparison between the two tools highlights, once again, the importance of the reference BED file used in determining coverage metrics. While the results are largely consistent across most coverage thresholds, notable differences, particularly in the Average Read Depth and the highest coverage levels, underscore the impact of BED file selection on the analysis results.

3.3 Performance

3.4 Users feedback

In order to evaluate the usability and effectiveness of the developed software, a feedback questionnaire was designed and distributed to a group of 14 users from Unilabs Genetics team with diverse professional backgrounds. The primary objective of this questionnaire was

to gather insights into the user experience, performance, and overall satisfaction with the software. Given that the software was intended to cater to a wide range of technical expertise, from laboratory technicians to bioinformatics specialists, it was crucial to ensure that it met the functional and usability needs of these varied audiences.

The questionnaire consisted of multiple sections aimed at assessing specific aspects of the software, including the ease of navigation, clarity of the interface, speed of performance, and the usefulness of the analytical metrics provided. Participants were asked to rate their experience on a numerical scale while also providing qualitative feedback on areas where the software excelled or required improvement.

In addition to rating the software's technical performance, respondents were encouraged to share their thoughts on the software's design, user interface, and the clarity of instructions provided throughout the analysis process. This approach allowed for a comprehensive evaluation that encompassed both the functional and aesthetic dimensions of the user experience. The ultimate goal of this survey was to identify key areas that could be optimized to further align the software with the needs of its users, while also validating its current performance and utility in the field of genomic analysis.

Overall, the feedback was overwhelmingly positive. Most users reported a high level of satisfaction with the software's intuitive interface, with many describing it as "very intuitive" and "user-friendly." All respondents confirmed that they did not experience any difficulties navigating the software. Additionally, the explanations provided for the functionalities were considered clear by all users, indicating that the design and guidance within the software successfully facilitated a smooth user experience.

In terms of performance, the software was well-regarded for its speed and efficiency. The majority of users rated the software's performance highly (considering the test dataset), with ratings ranging from 3 to 5 on a scale of 1 to 5, with 5 representing the fastest performance. The respondents also praised the ease with which they could input data and obtain analysis results, with all participants stating that the process was straightforward. This indicates that the software effectively supports the users workflow, minimizing potential barriers to data input and result interpretation.

The analysis results themselves were deemed clear and comprehensible, with users unanimously agreeing that the software met or exceeded their expectations. Notably, the metrics provided by the software were highlighted as particularly useful for their work, suggesting that the software successfully addresses key analytical needs in genomic analysis. Users expressed a high likelihood of recommending the software to colleagues, demonstrating strong approval of its functionality and usefulness.

Suggestions for improvement were minimal, with a few users recommending performance enhancements and the inclusion of additional features, such as tailoring the software for CNVs analysis. Nevertheless, users appreciated the versatility of the software, particularly the ability to zoom into specific regions and generate metrics for gene panels, genes and exons.

The feedback from the questionnaire indicates that the software is well-received, providing a user-friendly, intuitive, and efficient tool that meets the needs of various professionals in the

genomic field. The overwhelmingly positive responses suggest that the software has significant potential for widespread adoption, with only minor adjustments needed to further enhance its performance and expand its capabilities.

CHAPTER 4

Additional activities during the internship

"Always deliver more than expected." - Larry Page, co-founder of Google

4.1 Additional Activities during the Internship

As part of the internship, an additional activity involved the development of a gene panel creation tool. The purpose of the tool is to streamline the creation and management of gene panels, which are commonly used in genomic analyses.

The gene panel creator allows the user to define a panel by specifying a name and providing a list of genes. The user can paste the list of genes into the designated field, and the tool processes this input to create a gene panel. Once created, the panel becomes available within the system for further use in various analyses.

An additional feature of the tool is the generation of a BED file containing the genomic coordinates of the genes included in the panel. This file can be downloaded directly from the tool's interface, ensuring that the user has access to the relevant genomic regions for further study. The BED file is automatically verified to ensure that all genes are correctly recognized by the system before it is made available for download.

This gene panel creation tool was developed to improve the efficiency of handling gene panels, reducing the manual workload typically involved in their curation and preparation for analysis. The ability to automatically generate and download BED files associated with specific gene panels has proven to be a valuable addition to the software's functionality, ultimately benefiting the genomic analysis workflows at Unilabs.

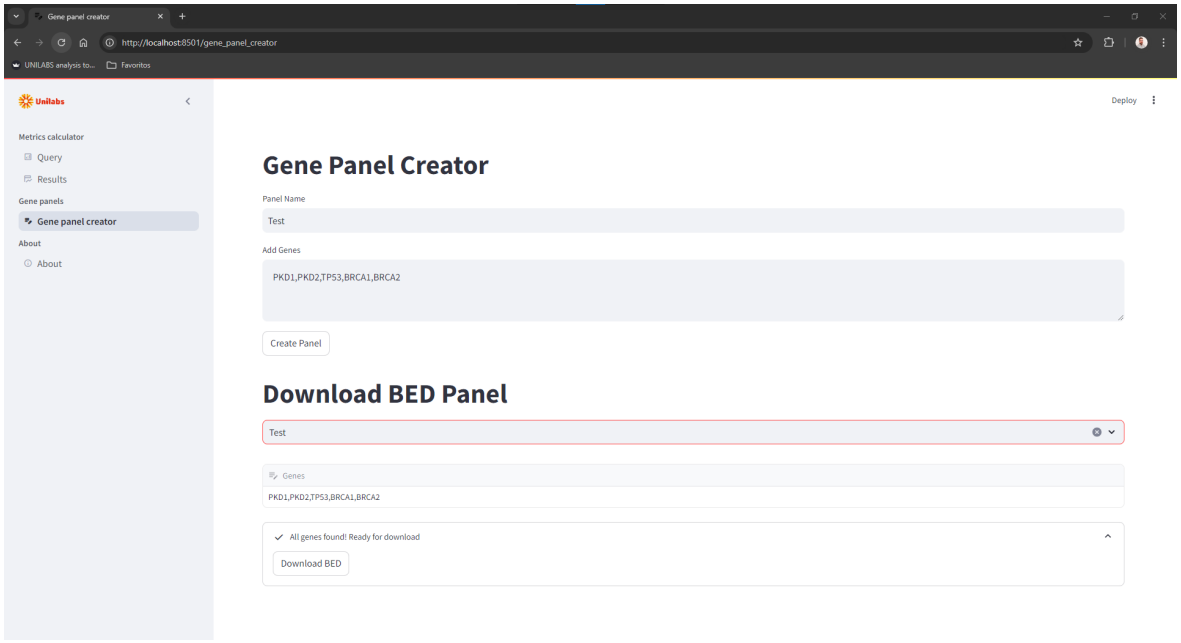


Figure 4.1: Gene Panel Creator Interface

CHAPTER 5

Discussion

"Discussion is an exchange of knowledge; an argument an exchange of ignorance." - Robert Quillen, humorist and journalist.

This chapter delves into a critical evaluation of the genomic analysis software developed during this project. It explores how the software stands within the broader landscape of bioinformatics and genomics, providing insights into its potential future directions. Additionally, the chapter discusses key optimizations undertaken to improve performance and efficiency, alongside the broader impact the software may have on the company. By analyzing both the internal and external factors affecting the software's development, this discussion highlights the strategic decisions made and their implications for future growth and refinement.

5.1 SWOT Analysis

During the development of the genomic analysis software, a Strengths Weeknesses Opportunities and Threats (SWOT) analysis was conducted to evaluate its strategic position in the field of bioinformatics and genomics. This analysis provided a comprehensive overview of the internal strengths and weaknesses, as well as the external opportunities and threats. The insights gained from this process helped guide the development and deployment of the software. The following subsections detail each component of the SWOT analysis.

5.1.1 Strengths

One of the major strengths of the software lies in its foundation on advanced bioinformatics and genomic concepts. The software was designed to meet critical genomic analysis needs, specifically focusing on Depth of Coverage and Breadth of Coverage, which are key metrics in NGS compliance. By addressing these metrics, the software ensures high-quality, reliable analysis, adhering to established bioinformatics guidelines.

Another strength is the utilization of modern development tools such as WSL, Anaconda, Conda, Git, GitHub, Streamlit, Docker and Python. These technologies not only provide a

robust and efficient development environment but also facilitate the software's deployment and use across different systems. Additionally, the software boasts a user-friendly interface that makes it accessible even to users without specialized bioinformatics expertise. This accessibility ensures that the tool can be used by a wider audience, including clinicians and lab technicians.

The use of Git and GitHub enhances the software's collaborative potential, ensuring traceability and reproducibility of code. These platforms allow for efficient version control and enable others to contribute to the project. Moreover, the use of Anaconda and Conda ensures that the software is deployed in an isolated and reproducible environment, where package and dependency management is handled efficiently, minimizing compatibility issues across systems.

5.1.2 Weaknesses

Despite its strengths, the software has several limitations. One significant weakness is its reliance on WSL and a Linux-based environment. This requirement may limit its accessibility for users unfamiliar with Linux systems, especially those who predominantly work in Windows or MacOS environments. While WSL enables Windows users to access a Linux environment, its setup and use may still be challenging for some.

Another notable weakness is related to the software's performance on conventional computers. While it can handle smaller panels with a relatively low number of genes, more extensive analyses, such as those involving data fromWES or large gene panels, require considerably more computational power. On a standard desktop or laptop, the processing time for such large datasets can become impractically long. For more extensive panels or whole-exome analyses, the software would benefit from parallel processing on a cluster or the use of distributed computing tools such as Dask [54]. These technologies can significantly reduce processing times, making the analysis of large-scale datasets more feasible within a reasonable timeframe.

Although the graphical interface is intuitive, it may not meet the needs of advanced users who prefer command-line tools for performing complex genomic analyses. This limitation could result in a divide between novice and expert users, with the latter potentially favoring other tools that offer more control and flexibility.

Another challenge lies in the initial setup of the development environment. The process involves multiple steps, including setting up the necessary tools and dependencies, which may present a barrier for users with less technical expertise. This complexity could deter adoption by some users, especially those with limited experience in software development and bioinformatics.

Lastly, while the software provides essential metrics, comparing and validating its outputs with those of other established tools can be difficult. Ensuring consistency and accuracy across different platforms is crucial for gaining user trust, but achieving this can be challenging due to variations in algorithms and methodologies used by different tools.

5.1.3 Opportunities

The software presents several growth opportunities. One of the most promising is its potential for expansion. The tool can be extended to include additional genomic analysis metrics beyond sequencing depth, providing a more comprehensive solution for genomic researchers and clinicians. Expanding the software's capabilities would make it even more valuable for those in the bioinformatics field.

Furthermore, the software could become a standard internal tool at Unilabs, gaining wider adoption across the organization for genomic metric analysis. This would reinforce its utility in real-world applications and provide valuable feedback for continuous improvement.

Another opportunity lies in the possibility of releasing the software as an open-source project. By doing so, a broader community of developers and users could contribute to its development, ensuring continuous updates, new features, and improvements. Open-sourcing the project could also increase its visibility and credibility within the bioinformatics community.

The rapid advancements in sequencing technologies and data analysis also present an opportunity to continuously enhance the software. New algorithms and methodologies are frequently developed in genomics, and staying updated with these trends could help the software remain relevant and cutting-edge. Regular updates based on the latest developments could position the software as a leading tool in genomic analysis.

5.1.4 Threats

Despite the opportunities, the software faces several external threats. One of the primary threats is the presence of established tools and platforms that offer similar functionalities. Competing with these well-established solutions may hinder the adoption of the new software, especially if the alternatives are more widely known or have more extensive support.

Additionally, commercial software solutions often provide comprehensive support, making it difficult for independent projects to compete. These commercial tools typically have larger development teams and resources, allowing them to quickly address issues, introduce new features, and provide user support, which could make adoption of the new software less appealing.

The rapid pace of evolution in sequencing technologies poses another threat. New advancements could quickly render some of the software's features obsolete or necessitate frequent updates to keep the tool current. The risk of obsolescence is especially high in bioinformatics, where the field is continuously advancing.

Moreover, the software depends on third-party libraries and tools, which introduces the risk of dependency-related issues. If one of the critical libraries or tools is discontinued or undergoes significant changes to its API, it could disrupt the software's functionality and require substantial rework.

Lastly, increasing regulation around genomic data and genetic testing could impose additional challenges for the software's use and distribution. Stringent data protection and privacy laws, particularly in the context of genetic information, could limit the software's adoption, especially in regions with more rigorous regulatory environments.

Several optimizations have been identified that could enhance its performance, efficiency, and scalability. Although these optimizations have not yet been implemented, they represent critical next steps for future versions of the software.

5.2.1 Parallel and Distributed Processing with Dask

One of the most significant opportunities for optimization is the integration of parallel and distributed processing using Dask. Currently, the software processes data sequentially, which limits its scalability, especially when handling large datasets such as genomic sequences. By leveraging Dask, the software could distribute workloads across multiple CPU cores or even multiple machines, dramatically reducing execution time and allowing for greater scalability. Dask's ability to handle out-of-core computation would also be beneficial for memory-intensive operations, making the system more efficient when processing large data files.

5.2.2 Integration with AWS S3 Storage Service

Another key optimization involves the implementation of a connection with the AWS S3 data storage service of Unilabs. By integrating with AWS S3, the software could leverage cloud-based storage for genomic data, enabling faster access, improved data security, and enhanced scalability. The BAM/CRAM files could be acessed and used directly from the cloud, eliminating the need for local storage and reducing the burden on users systems.

5.2.3 Improvements to the Graphical User Interface

The software's GUI could also benefit from significant optimization. Presently, the user experience is functional but lacks the fluidity and responsiveness necessary for efficient navigation. Implementing more dynamic and responsive design principles would provide a smoother user experience, particularly when interacting with large datasets or complex visualizations. Furthermore, enhancing the layout to support multi-threaded operations in the backend would ensure that the interface remains responsive even during heavy computation.

5.2.4 Secure and Efficient Authentication and Authorization System

If the software is to be deployed in a production environment, implementing a secure authentication and authorization system is essential. Currently, the software lacks robust user authentication mechanisms, making it vulnerable to unauthorized access and data breaches. By integrating secure authentication protocols such as OAuth or OpenID Connect, the software could ensure that only authorized users can access sensitive data and functionalities, namely the data stored in the AWS S3 storage service.

5.2.5 Enhanced Software Documentation

Improving software documentation is another area that requires attention. While the current documentation provides basic guidelines for installation and usage, it lacks comprehensive instructions for developers and users, particularly regarding complex functionalities.

Enhancing the documentation to include more detailed explanations of the codebase, installation procedures, and advanced usage scenarios would make the software more accessible to both new users and contributors. Clear and structured documentation would also facilitate future development efforts by providing a solid foundation for understanding the software's architecture and functionality.

5.2.6 Implementation of Automated Testing

The introduction of automated testing is a crucial step toward improving the software's reliability and maintainability. Currently, testing is performed manually, which can be time-consuming and prone to error. Implementing automated unit tests, integration tests, and end-to-end tests would ensure that the software functions as expected after updates or modifications. Automated testing would also help identify bugs early in the development process, reducing the risk of regressions and ensuring that the software remains robust as new features are added.

5.2.7 Code Optimization for Efficiency and Execution Time

Optimizing the code to improve execution efficiency and reduce runtime is another significant potential enhancement. Although the software performs adequately for smaller datasets, its performance degrades when processing large-scale genomic data. Refactoring the code to eliminate bottlenecks, streamline algorithms, and reduce memory consumption would enhance the software's performance. Additionally, optimizing the use of external libraries such as SAMtools would further reduce execution times, ensuring that the software can handle large datasets more efficiently.

5.2.8 Implementation of Monitoring and Alert Systems

Finally, implementing a monitoring and alert system would enable proactive detection of performance issues and anomalies. Currently, the software lacks real-time monitoring, which limits the ability to detect and resolve issues as they arise. By integrating a monitoring system, the software would be able to track key performance metrics such as memory usage, CPU load, and execution times. Additionally, an alert system could notify administrators of critical issues, allowing for faster resolution and ensuring that the software remains operational and efficient under various conditions.

Although these optimizations have yet to be implemented, they represent a roadmap for future development and will be essential for ensuring that the software can meet the demands of large-scale genomic analysis.

5.2.9 Impact of Matched Annotation from NCBI and EMBL-EBI Transcripts on Software Metrics

The MANE project is a collaborative initiative that provides standardized transcript annotations for human protein-coding genes. Each MANE transcript is carefully selected to exactly match between RefSeq and Ensembl/GENCODE annotations, ensuring consistent use across both databases. [51]

The integration of MANE transcripts into the software was a key enhancement that standardized gene and exon annotations across multiple sources, specifically harmonizing RefSeq and Ensembl/GENCODE annotations. This unification was crucial for establishing consistency in the calculation of metrics such as Depth of Coverage and Breadth of Coverage. By utilizing a single, universally recognized transcript set, the software minimized the variability caused by discrepancies in different annotations. The selection of the most biologically relevant MANE transcript for each protein-coding gene ensured that coverage analyses were grounded in a robust and widely accepted reference.

Furthermore, the adoption of MANE transcripts brought the software in line with the global trend towards standardized genomic annotations in both research and clinical settings. This alignment not only enhanced the reliability of key metrics but also improved the accuracy and consistency of the software's analyses, making it more effective for both genomic research and clinical diagnostics. The standardized annotations ensure that the software produces dependable results, which are crucial for precise interpretation and decision-making in these fields.

5.3 Impact on the company

The overwhelmingly positive feedback from the Unilabs Genetics team demonstrates that the developed software has the potential to make a significant impact within the organization. By providing an intuitive, user-friendly interface, the software successfully addresses the key analytical needs of various professionals, from laboratory technicians to bioinformatics specialists. This broad applicability suggests that the software could streamline workflows, enhance productivity, and reduce the time required to perform complex genomic analyses.

One of the most notable impacts of the software lies in its ability to simplify data processing and result interpretation. Users highlighted the software's capacity to present analysis results in a clear and understandable manner, which directly contributes to reducing potential bottlenecks in the workflow. This aspect is particularly valuable in a high-throughput environment like Unilabs, where rapid and accurate analysis of genomic data is critical. The ease with which users can input data and obtain meaningful insights reinforces the software's role in facilitating more efficient operational processes.

The versatility of the software is another key factor in its potential impact. Users praised the flexibility of the tool, particularly its ability to handle gene panels, single gene analyses, and exon-level metrics. This flexibility not only aligns with the diverse needs of the Unilabs Genetics team but also enhances the software's capacity to support different levels of genomic analyses.

While the software's current performance was well-received for test data, there were some suggestions for further improvements for dealing with bigger datasets. These suggestions highlight the software's potential for growth and its ability to adapt to evolving analytical needs.

The impact of this software on Unilabs is not solely limited to the immediate improvements in operational efficiency and user satisfaction. The successful implementation of this tool also

has strategic implications for Unilabs. By utilizing an internally developed software solution tailored to their specific needs, Unilabs can reduce dependency on external software providers and improve control over their analytical workflows. Furthermore, the internal development and continuous improvement of such tools foster innovation within the organization, enhancing Unilabs competitive edge in the field of genomic diagnostics.

CHAPTER 6

Final remarks

"When something is important enough, you do it even if the odds are not in your favor." - Elon Musk

This work has successfully resulted in the development of a tool designed to capture essential metrics for the quality assessment of NGS data. The tool was built with a focus on providing researchers and laboratory technicians with a fast and efficient way to evaluate the quality of NGS data. Despite certain limitations, particularly in terms of performance on conventional computing systems, this tool has demonstrated significant potential for further development. By evolving the tool to be compatible with high-performance computing environments, it could be optimized to handle the large volumes of data typically associated with NGS analysis, thereby extending its utility and scope.

Throughout this project, several challenges were faced, most notably the performance constraints of the software on standard computing hardware. However, these limitations also offer a clear path for future optimizations and enhancements. The prospect of integrating the software into high-performance computing platforms suggests that with further development, the tool could be adapted to meet the demands of large-scale genomic analysis more effectively. Such advancements would enable the tool to become a valuable asset not only in research contexts but also in clinical and industrial applications where the rapid and accurate evaluation of sequencing data is critical.

This internship has significantly contributed to my growth as a bioinformatician, equipping me with a deeper understanding of bioinformatics and genomics. Engaging with experts in these fields provided me with invaluable knowledge and practical experience, which will undoubtedly be beneficial in my professional future. Moreover, this experience has allowed me to bridge the gap between academic knowledge and its real-world application. The hands-on nature of this project facilitated the application of concepts learned throughout my studies while simultaneously fostering the development of key skills essential for my future career.

In conclusion, this internship has been both enriching and challenging, offering a unique opportunity to contribute to the development of a practical tool that addresses critical needs

in genomic data analysis. The experience has not only solidified my expertise in bioinformatics but also opened new avenues for future exploration in the realm of high-performance genomic analysis. As the tool continues to evolve and expand, it holds the promise of becoming an indispensable resource for researchers and clinicians alike, paving the way for faster, more accurate evaluations of NGS data.

Bibliography

[1] *Unilabs - sobre*. [Online]. Available: https://www.unilabs.pt/pt/a-unilabs/sobre-nos/unilabs-portugal.

[2] *Unilabs - genética médica*. [Online]. Available: https://www.unilabs.pt/pt/servicos/especialidades-medicos/genetica-medica/sobre.

[3] N. H. G. R. I. (NHGRI), *Genetic timeline*. [Online]. Available: https://www.genome.gov/Pages/Education/GeneticTimeline.pdf.

[4] J. Gayon, «De mendel à l'épigénétique: Histoire de la génétique», *Comptes Rendus - Biologies*, vol. 339, pp. 225–230, 7-8 Jul. 2016, ISSN: 17683238. DOI: 10.1016/j.crvi.2016.05.009.

[5] F. S. Collins and L. ; Fink, *The human genome project*, 1995.

[6] M. Jinek, K. Chylinski, I. Fonfara, M. Hauer, J. A. Doudna, and E. Charpentier, *A programmable dual-rna-guided dna endonuclease in adaptive bacterial immunity*. [Online]. Available: https://www.science.org.

[7] M. A. Gutierrez-Reinoso, P. M. Aponte, and M. Garcia-Herreros, *Genomic analysis, progress and future perspectives in dairy cattle selection: A review*, Mar. 2021. DOI: 10.3390/ani11030599.

[8] N. H. G. R. Institute, *Genetics vs. genomics fact sheet*, Sep. 2018. [Online]. Available: https://www.genome.gov/about-genomics/fact-sheets/Genetics-vs-Genomics.

[9] T. J. Laboratory, *Genetics vs. genomics*, Feb. 2017. [Online]. Available: https://www.jax.org/personalized-medicine/precision-medicine-and-you/genetics-vs-genomics#.

[10] S. Minchin and J. Lodge, *Understanding biochemistry: Structure and function of nucleic acids*, 2019. DOI: 10.1042/EBC20180038.

[11] M. KGaA, *Sanger sequencing steps and method*. [Online]. Available: https://www.sigmaaldrich.com/PT/en/technical-documents/protocol/genomics/sequencing/sanger-sequencing.

[12] S. M. Group, *Crick and watson's dna molecular model*, 1977. [Online]. Available: https://collection.sciencemuseumgroup.org.uk/objects/co146411/crick-and-watsons-dna-molecular-model..

[13] B. Maddox, *The double helix and the 'wronged heroine'*, Jan. 2003. DOI: 10.1038/nature01399.

[14] L. Merrick, A. Campbell, D. Muenchrath, and S. Fei., «Mutations and variation», in W. Suza and K. Lamkey, Eds. Iowa State University Digital Press, Mar. 2016. DOI: 10.31274/isudp.2023.130.

[15] C. A. for Drugs and T. in Health, *Next generation dna sequencing: A review of the cost effectiveness and guidelines*, Feb. 2014.

[16] H. L. Rehm, S. J. Bale, P. Bayrak-Toydemir, *et al.*, «Acmg clinical laboratory standards for next-generation sequencing», *Genetics in Medicine*, vol. 15, pp. 733–747, 9 Sep. 2013, ISSN: 10983600. DOI: 10.1038/gim.2013.92.

[17] J. Majewski, J. Schwartzentruber, E. Lalonde, A. Montpetit, and N. Jabado, «What can exome sequencing do for you?», *Journal of Medical Genetics*, vol. 48, no. 9, pp. 580–589, 2011, ISSN: 0022-2593. DOI: 10.1136/jmedgenet-2011-100223. eprint: https://jmg.bmj.com/content/48/9/580.full.pdf. [Online]. Available: https://jmg.bmj.com/content/48/9/580.

[18] N. J. Schork, «Genetic parts to a preventive medicine whole», *Genome Medicine*, vol. 5, no. 6, p. 54, Jun. 2013, ISSN: 1756-994X. DOI: 10.1186/gm458. [Online]. Available: https://doi.org/10.1186/gm458.

[19] T. C. GLENN, «Field guide to next-generation dna sequencers», *Molecular Ecology Resources*, vol. 11, no. 5, pp. 759–769, 2011. DOI: https://doi.org/10.1111/j.1755-0998.2011.03024.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1755-0998.2011.03024.x. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1755-0998.2011.03024.x.

[20] Illumina, *Novaseq 6000 sequencing system guide*, Feb. 2023. [Online]. Available: https://emea.support.illumina.com/downloads/novaseq-6000-system-guide-1000000019358.html.

[21] N. B. Larson, A. L. Oberg, A. A. Adjei, and L. Wang, *A clinician's guide to bioinformatics for next-generation sequencing*, Feb. 2023. DOI: 10.1016/j.jtho.2022.11.006.

[22] Wikipedia, *Fastq format*, Jun. 2024. [Online]. Available: https://en.wikipedia.org/wiki/FASTQ_format.

[23] S. Roy, C. Coldren, A. Karunamurthy, *et al.*, *Standards and guidelines for validating next-generation sequencing bioinformatics pipelines: A joint recommendation of the association for molecular pathology and the college of american pathologists*, Jan. 2018. DOI: 10.1016/j.jmoldx.2017.11.003.

[24] M. Bioinformatics, *Structural variant calling - long read data.* [Online]. Available: https://www.melbournebioinformatics.org.au/tutorials/tutorials/longread_sv_calling/longread_sv_calling/.

[25] R. Somak, *Next-generation sequencing bioinformatics pipelines*, Mar. 2020. [Online]. Available: https://www.myadlm.org/cln/Articles/2020/March/Next-Generation-Sequencing-Bioinformatics-Pipelines.

[26] A. M. Kanzi, J. E. San, B. Chimukangara, *et al.*, «Next generation sequencing and bioinformatics analysis of family genetic inheritance», *Frontiers in Genetics*, vol. 11, 2020, ISSN: 1664-8021. DOI: 10.3389/fgene.2020.544162. [Online]. Available: https://www.frontiersin.org/journals/genetics/articles/10.3389/fgene.2020.544162.

[27] N. M. Ioannidis, J. H. Rothstein, V. Pejaver, *et al.*, «Revel: An ensemble method for predicting the pathogenicity of rare missense variants», *American Journal of Human Genetics*, vol. 99, pp. 877–885, 4 Oct. 2016, ISSN: 15376605. DOI: 10.1016/j.ajhg.2016.08.016.

[28] M. Schubach, T. Maass, L. Nazaretyan, S. Roner, and M. Kircher, «Cadd v1.7: Using protein language models, regulatory cnns and other nucleotide-level scores to improve genome-wide variant predictions», *Nucleic Acids Research*, vol. 52, pp. D1143–D1154, D1 Jan. 2024, ISSN: 13624962. DOI: 10.1093/nar/gkad989.

[29] Y. Fu, Z. Liu, S. Lou, *et al.*, «Funseq2: A framework for prioritizing noncoding regulatory variants in cancer», *Genome biology*, vol. 15, p. 480, 10 2014, ISSN: 1474760X. DOI: 10.1186/s13059-014-0480-5.

[30] A. P. Boyle, E. L. Hong, M. Hariharan, *et al.*, «Annotation of functional variation in personal genomes using regulomedb», *Genome Research*, vol. 22, pp. 1790–1797, 9 Sep. 2012, ISSN: 10889051. DOI: 10.1101/gr.137323.112.

[31] I. Dunham, A. Kundaje, S. F. Aldred, *et al.*, «An integrated encyclopedia of dna elements in the human genome», *Nature*, vol. 489, pp. 57–74, 7414 Sep. 2012, ISSN: 14764687. DOI: 10.1038/nature11247.

[32] R. E. Consortium, A. Kundaje, W. Meuleman, *et al.*, «Integrative analysis of 111 reference human epigenomes», *Nature*, vol. 518, pp. 317–329, 7539 Feb. 2015, ISSN: 14764687. DOI: 10.1038/nature14248.

[33] S. Chen, L. C. Francioli, J. K. Goodrich, *et al.*, «A genomic mutational constraint map using variation in 76,156 human genomes», *Nature*, vol. 625, pp. 92–100, 7993 Jan. 2024, ISSN: 14764687. DOI: 10.1038/s41586-023-06045-0.

[34] M. J. Landrum, J. M. Lee, M. Benson, *et al.*, «Clinvar: Improving access to variant interpretations and supporting evidence», *Nucleic Acids Research*, vol. 46, pp. D1062–D1067, D1 Jan. 2018, ISSN: 13624962. DOI: 10.1093/nar/gkx1153.

[35] P. D. Stenson, M. Mort, E. V. Ball, *et al.*, *The human gene mutation database (hgmd®): Optimizing its use in a clinical diagnostic or research setting*, Oct. 2020. DOI: 10.1007/s00439-020-02199-3.

[36] J. G. Tate, S. Bamford, H. C. Jubb, *et al.*, «Cosmic: The catalogue of somatic mutations in cancer», *Nucleic Acids Research*, vol. 47, pp. D941–D947, D1 Jan. 2019, ISSN: 13624962. DOI: 10.1093/nar/gky1015.

[37] K. Wang, M. Li, and H. Hakonarson, «Annovar: Functional annotation of genetic variants from high-throughput sequencing data», *Nucleic Acids Research*, vol. 38, 16 Jul. 2010, ISSN: 03051048. DOI: 10.1093/nar/gkq603.

[38] 3billion, *Sequencing depth vs coverage*, Aug. 2023. [Online]. Available: https://3billion.io/blog/sequencing-depth-vs-coverage.

[39] MedGenome, *Understanding gene coverage and read depth*, Apr. 2020.

[40] G. for Geeks, *Functional vs non functional requirements*, Jun. 2024. [Online]. Available: https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/.

[41] M. 2024, *How to install linux on windows with wsl*. [Online]. Available: https://learn.microsoft.com/en-us/windows/wsl/install.

[42] A. Inc, *Anaconda*. [Online]. Available: https://docs.anaconda.com/free/.

[43] J. Leidel, *12 reasons to choose conda*, Sep. 2023. [Online]. Available: https://www.anaconda.com/blog/12-reasons-to-choose-conda.

[44] A. Inc, *Anaconda - installing on windows*. [Online]. Available: https://docs.anaconda.com/free/anaconda/install/windows/.

[45] A. Inc, *Anaconda - managing environments*. [Online]. Available: https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html#create-env-file-manually.

[46] G. Inc, «Git and github - get started», [Online]. Available: https://docs.github.com/pt/get-started/start-your-journey.

[47] A. Sehm, *Introduction to streamlit and streamlit components*, Apr. 2022. [Online]. Available: https://auth0.com/blog/introduction-to-streamlit-and-streamlit-components/.

[48] Dayanithi, *Streamlit in 3 minutes*, Apr. 2023. [Online]. Available: https://medium.com/data-and-beyond/streamlit-d357935b9c.

[49] Streamlit, «Api reference», 2024. [Online]. Available: https://docs.streamlit.io/develop/api-reference.

[50] R. J. Kinsella, A. K. H. Ri, S. Haider, *et al.*, «Ensembl biomarts: A hub for data retrieval across taxonomic space», DOI: 10.1093/database/bar030. [Online]. Available: https://academic.oup.com/database/article/doi/10.1093/database/bar030/465356.

[51] J. Morales, S. Pujar, J. E. Loveland, *et al.*, «A joint ncbi and embl-ebi transcript set for clinical genomics and research», *Nature*, 2022, ISSN: 14764687. DOI: 10.1038/s41586-022-04558-8.

[52] Ensembl, *Tark - transcript archive*. [Online]. Available: https://tark.ensembl.org/.

[53] D. Merkel, «Docker: Lightweight linux containers for consistent development and deployment», *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[54] Dask Development Team, *Dask: Library for dynamic task scheduling*, 2016. [Online]. Available: http://dask.pydata.org.

[55] B. .-. Illumina, *Quality score encoding*, May 2024. [Online]. Available: https://help.basespace.illumina.com/files-used-by-basespace/quality-scores.

APPENDIX A

Additional content

Table A.1: Unilabs test catalog

Test Catalog
WES
Next Generation Sequencing
Sanger Sequencing
Comparative Genomic Hybridization (aCGH)
Karyotyping
Fluorescence In Situ Hybridization (FISH)
QF-PCR, qPCR, RT-PCR
Fragment and Expansion Analysis
Multiplex Ligation-Dependent Probe Amplification (MLPA)
Single Gene Analysis
Variant Analysis
Cytogenetics
NIPT | Tomorrow

Table A.2: Quality score encoding. Adapter from [55]

Symbol	ASCII Code	Q-Score	P-Error
!	33	0	1,00000
"	34	1	0,79433
#	35	2	0,63096
$	36	3	0,50119
%	37	4	0,39811
&	38	5	0,31623
’	39	6	0,25119
(	40	7	0,19953
)	41	8	0,15849
*	42	9	0,12589
+	43	10	0,10000
,	44	11	0,07943
-	45	12	0,06310
.	46	13	0,05012
/	47	14	0,03981
0	48	15	0,03162
1	49	16	0,02512
2	50	17	0,01995
3	51	18	0,01585
4	52	19	0,01259
5	53	20	0,01000
6	54	21	0,00794
7	55	22	0,00631
8	56	23	0,00501
9	57	24	0,00398
:	58	25	0,00316
;	59	26	0,00251
<	60	27	0,00200
=	61	28	0,00158
>	62	29	0,00126
?	63	30	0,00100
@	64	31	0,00079
A	65	32	0,00063
B	66	33	0,00050
C	67	34	0,00040
D	68	35	0,00032
E	69	36	0,00025
F	70	37	0,00020
G	71	38	0,00016
H	72	39	0,00013
I	73	40	0,00010

Table A.3: Packages used in the project

Package Name	Version
python	3.12.1
numpy	1.26.3
pandas	2.1.4
plotly	5.24.1
boto3	1.34.150
weasyprint	62.3
streamlit	1.38.0
openpyxl	3.1.2
cairo	1.18.0
pango	1.50.14
jpeg	-
gdk-pixbuf	-
pip	23.3.1