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.

# DOCUMENTO PROVISÓRIO

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.

# DOCUMENTO PROVISÓRIO

"*The greatest challenge to any thinker is stating the problem in a way that will allow a solution*"

— Bertrand Russell

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 minha esposa e filho pelo incansável apoio.

**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 toda a ajuda a todos os meus colegas e companheiros.

**Palavras Chave** NGS, arquitetura, história, construção, materiais de construção, saber tradicional.

**Resumo**

Um resumo é um pequeno apanhado de um trabalho mais longo (como uma tese, dissertação ou trabalho de pesquisa). O resumo relata de forma concisa os objetivos e resultados da sua pesquisa, para que os leitores saibam exatamente o que se aborda no seu documento.
Embora a estrutura possa variar um pouco dependendo da sua área de estudo, o seu resumo deve descrever o propósito do seu trabalho, os métodos que você usou e as conclusões a que chegou.
Uma maneira comum de estruturar um resumo é usar a estrutura IMRaD. Isso significa:

- Introdução
- Métodos
- Resultados
- Discussão

Veja mais pormenores aqui:
`https://www.scribbr.com/dissertation/abstract/`

**Keywords** textbook, architecture, history, construction, construction materials, traditional knowledge.

**Abstract**

An abstract is a short summary of a longer work (such as a thesis, dissertation or research paper).
The abstract concisely reports the aims and outcomes of your research, so that readers know exactly what your paper is about.
Although the structure may vary slightly depending on your discipline, your abstract should describe the purpose of your work, the methods you've used, and the conclusions you've drawn.
One common way to structure your abstract is to use the IMRaD structure. This stands for:

- Introduction
- Methods
- Results
- Discussion

Check for more details here:
`https://www.scribbr.com/dissertation/abstract/`

## 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

# List of Figures

# List of Tables

# List of Code Snippets

# 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
**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

CHAPTER 1

# Introduction

*"The only source of knowledge is experience." - Albert Einstein*

## 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.

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 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 Next Generation Sequencing (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. In addition, the depth of coverage directly influences the ability to detect genetic variants: regions with low 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 will be explained in detail below, the software created and described in this report allows the obtaining of Average Read Depth and Percentage 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 divided into five main chapters, each with several sections, and includes additional content at the end.

The first chapter, **Introduction**, begins with the **Internship Context and Framework**, providing an overview of the internship setting and its relevance. This is followed by a presentation of the **Project Motivation and Objectives**, outlining the purpose and goals of the work. The chapter also includes the **Document Structure**, this section, which explains how the document is organized. The **Characterization of the Host Entity and Work Plan** section describes the host entities, including Unilabs Portugal and Unilabs Genetics, and provides the timeline for the work. Finally, the **Theoretical Framework** section covers important concepts in Genetics and Genomics, Sequencing Methods and Characteristics, and Bioinformatics, including sub-sections on the evolution of genetics, the structure and function of DNA, the discovery of the double helix, DNA packaging, mutations, and genetic variations, as well as Next Generation Sequencing and a SWOT analysis.

The second chapter, **Software Development Process**, is detailed as follows: The **Analysis** section outlines the preliminary work, followed by **Planning**, which covers the project planning phase. The **Design** section describes the design phase, and **Development** covers the creation of the software, including **Environment Preparation**. The chapter also includes sections on **Testing and Validation**, **Optimization**, **Deployment**, **Documentation**, and **Maintenance**, explaining how each phase contributes to the software development lifecycle. This structure of software presentation is based on the SWEBOK Guide [1], which provides a comprehensive overview of software engineering practices.

The third chapter, **Additional Activities During the Internship**, highlights other activities and experiences gained during the internship.

The fourth chapter, **Discussion**, provides a comprehensive analysis of the work, interpreting results in the context of existing knowledge and discussing practical applications, limitations, and implications.

The fifth chapter, **Final Remarks**, presents conclusions drawn from the work and suggests possible improvements and new features for future versions of the software.

The document concludes with a **Bibliography** listing all consulted and cited sources, followed by Additional Content 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.

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.

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.

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. [2]

### 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.

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.

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. [3]

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 1.2 outlines the key milestones and deadlines that were met during this internship.

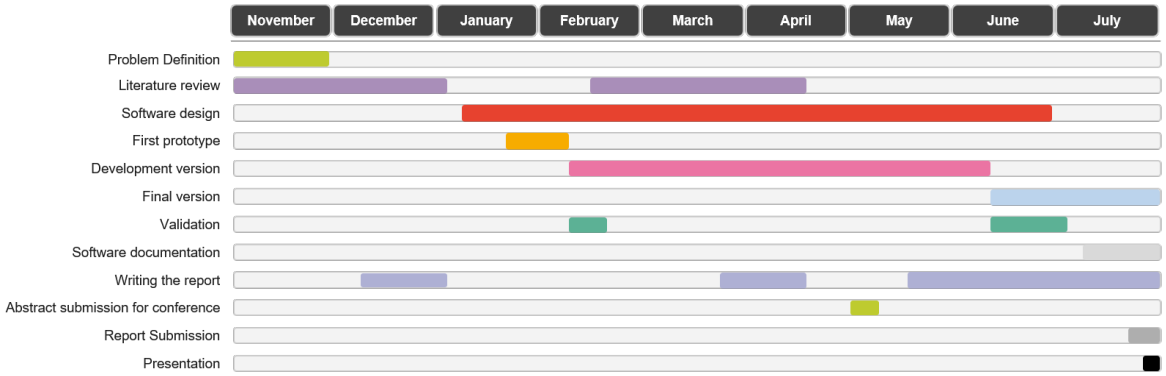



**Figure 1.1:** Imagem provisória do cronograma do estágio.

## 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. [4]

This understanding persisted until the 1950s, when DNA was discovered as the material basis of heredity, marking the start of molecular biology. [5]

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. [5]

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 [6], 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. [5]

The discovery of the Ribonucleic Acid (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. [7]

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

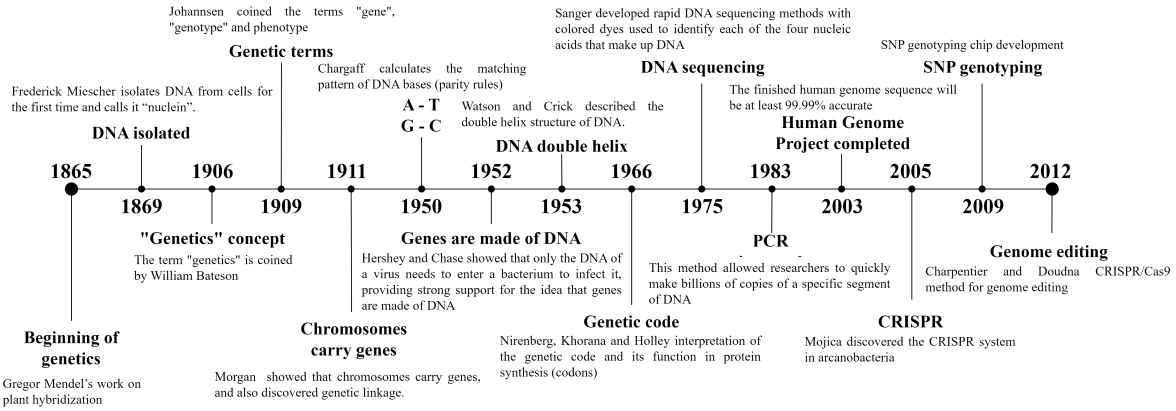



**Figure 1.2:** Evolution of genetics over the years: A brief timeline with some of the major historical milestones. Image adapted from [8] and [4]

### 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. [9]

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. [10]

### 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. [11]

### 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. [11]

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).

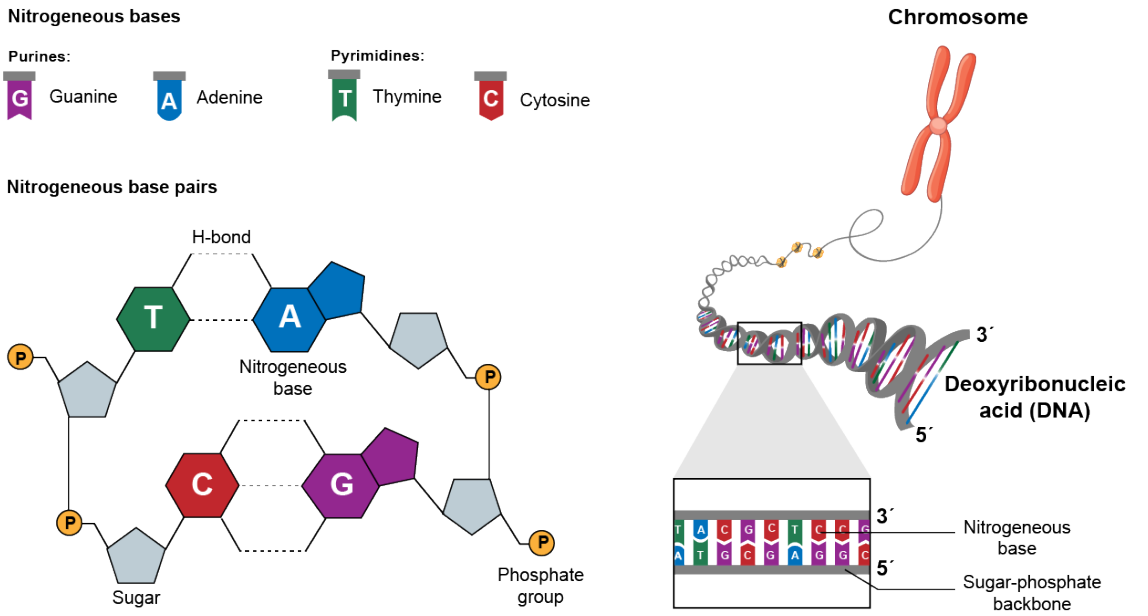
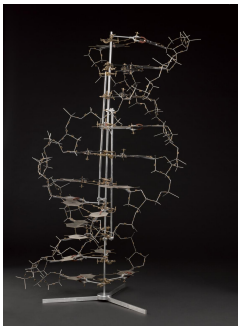


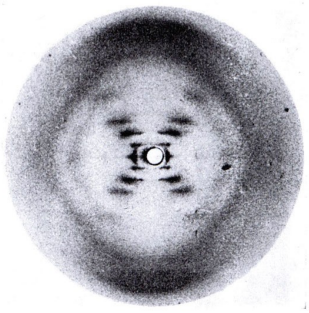

**Figure 1.3:** Representation of DNA and its constituents. In the top left corner, the nitrogenous bases are illustrated, categorized into Purines (Guanine and Adenine) and Pyrimidines (Thymine and Cytosine). Below this, on the left side, the paired nitrogenous bases are shown, along with their respective hydrogen bonds and the sugar-phosphate backbone connections. On the right side of the image, a chromosome is depicted unraveling, revealing the DNA structure. [12]

### 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. [11]



**(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 [13] 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 [14]

### 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. [11]

### 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. [15]

Genetic variation, on the other hand, 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. In cultivated plants, human-directed selection can also manipulate genetic variation to enhance desirable traits. [15]

## 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 [16], 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. [17]

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). [17]

#### *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. [17]

#### *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. [18] ES is particularly valuable in "detecting variants in known disease-associated genes as well as for the discovery of novel gene-disease associations" [17], with clinical studies demonstrating a diagnostic success rate of approximately 20%. [19] 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. [17]

#### *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. [17]

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 [17].

### 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. [17]

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. [17]

#### *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. [17]

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. [17]

#### *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. [17]

#### *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. [20] The choice of sequencing platform depends on various factors, including sequence capacity, read length, run time, cost, and accuracy. [17]

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. [17]

## 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. [17]

#### *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. [17]

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. [21]

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. [21]

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.

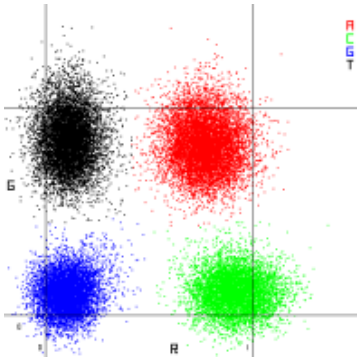



**Figure 1.5:** Visualization of cluster intensities in 2-Channel Sequencing (NovaSeq 6000 - Illumina). The image shows the intensity data for each cluster from both the red and green channels, with each cluster representing a different DNA base. Image from [21].

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 [21]

| 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. [22] 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).

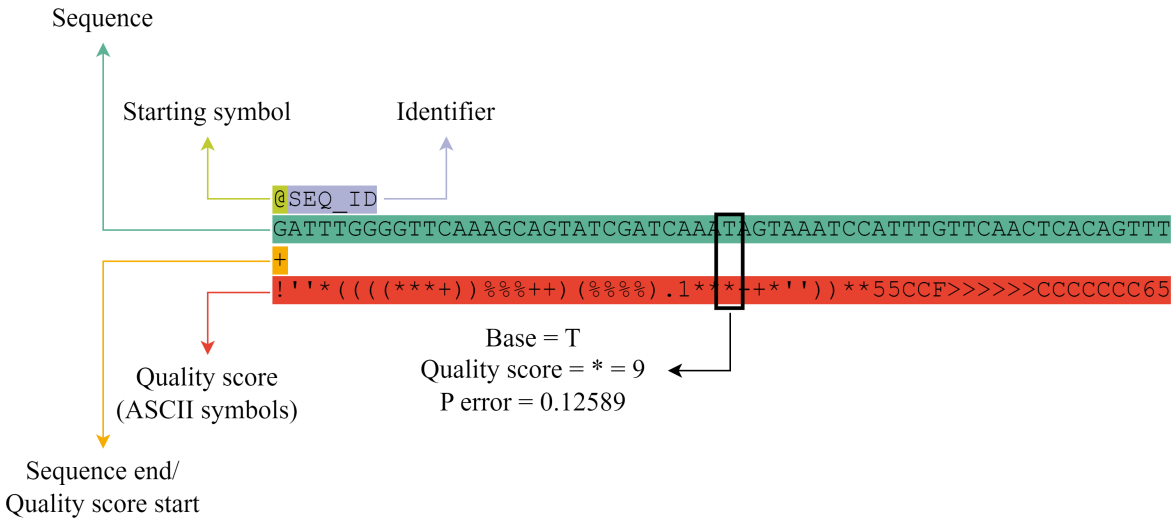



**Figure 1.6:** FASTQ file format example. The image shows a read identifier, nucleotide sequence, and quality score string in a FASTQ file. The P error calculation was performed based on the Phred quality score equation and Quality Score Encoding from Table A.2. This file example was adapted from [23].

Once the raw sequencing data was stored in a FASTQ file, the next step is **Read Alignment/Mapping**. [22] 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). [17] 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 Binary Alignment Map (BAM) formats, with BAM being more efficient due to compression and indexing capabilities. A more compressed format, Compressed Reference-oriented Alignment Map (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. [22]

### *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), single-nucleotide variants (SNVs), small insertions and deletions (INDELs), and copy number variants or alterations (CNVs and CNAs). The first tree are related to Sequence Variants, while the last one is related to Structural Variants. [24], [25], [26], [27] 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. [22] There are also other types of variants, such as translocations, inversions, and complex rearrangements, which are less common but can have significant clinical implications. [24]

The confidence with which variants are called depends heavily on sequencing depth 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. [22]

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. [22]

**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 variant allele frequencies (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. [22]

When detecting copy number variants (CNVs) using NGS data, algorithms primarily rely on sequencing coverage patterns, although they may also take into account the VAFs of overlapping variants. CNV 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, CNV identification within individual samples is difficult, and many tools designed for this purpose compare the data against a reference set of normal

samples. [22]

**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. [22]

Variants are typically stored in variant call format (VCF) files, which can include data from multiple samples. Genomic VCF (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. [22]

### *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. [17]

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 [28] predict the functional impact of missense variants, while noncoding variants are assessed using resources such as CADD [29], FunSeq2 [30], and RegulomeDB [31]. These are complemented by data from projects like ENCODE [32] and Roadmap Epigenomics [33], and external databases like gnomAD [34], ClinVar [35], HGMD [36], and COSMIC [37]. Annotation software like ANNOVAR [38] integrates these resources to enrich VCF files with variant details. [22]

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.

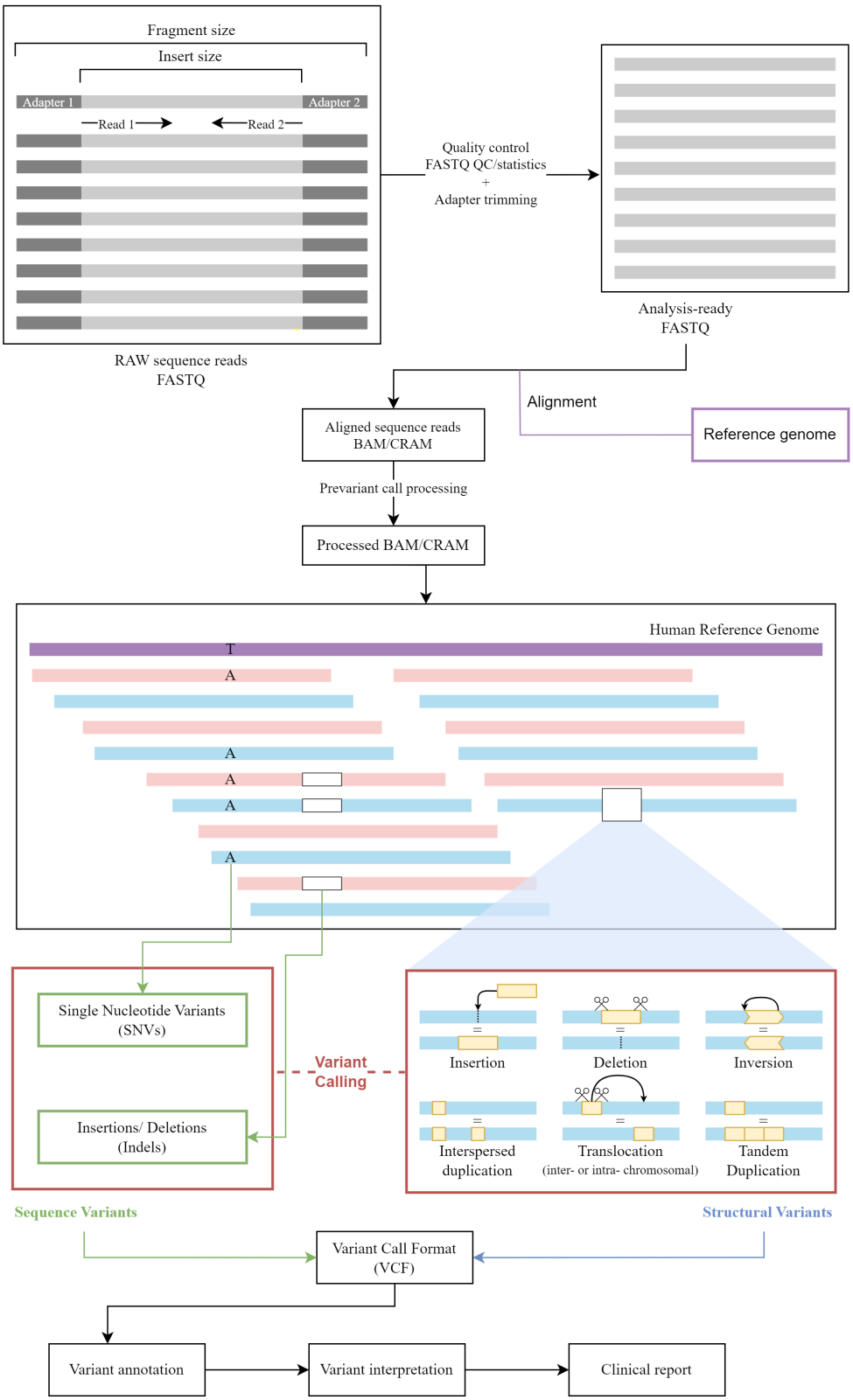



**Figure 1.7:** NGS data analysis pipeline. Adapted from [24], [25], [26], [27]

### 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 and sequencing depth. 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 coverage, which measures the proportion of the genome or target region that has been sequenced at least once. In technical documentation, coverage is often represented by a number followed by "X" such as 30X coverage. [22]

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 coverage at 30X would be expressed as "95% coverage." However, several factors can influence 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. [39]

Coverage is a fundamental metric in bioinformatics, as it directly impacts the reliability of detecting genetic variants. Higher coverage means that more sequencing reads overlap each base, leading to improved sensitivity and accuracy in identifying variations. In contrast, lower 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. [22]

Different types of sequencing require varying levels of coverage. For Whole Genome Sequencing (WES), coverage between 30X and 50X is generally recommended, while Whole Exome Sequencing (WES) typically calls for 100X coverage. In the case of targeted gene panels, the 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. [22]

#### *Sequencing/Read Depth or Depth of Coverage*

Sequencing depth, also referred to as read depth, 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 sequencing depth at that location would be expressed as 30X. [39]

The primary advantage of higher sequencing depth lies in its ability to enhance the precision of variant detection at specific genomic positions. This is particularly valuable when investigating rare genetic variants or when analyzing samples with significant heterogeneity, such as tumor tissues. [39]

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

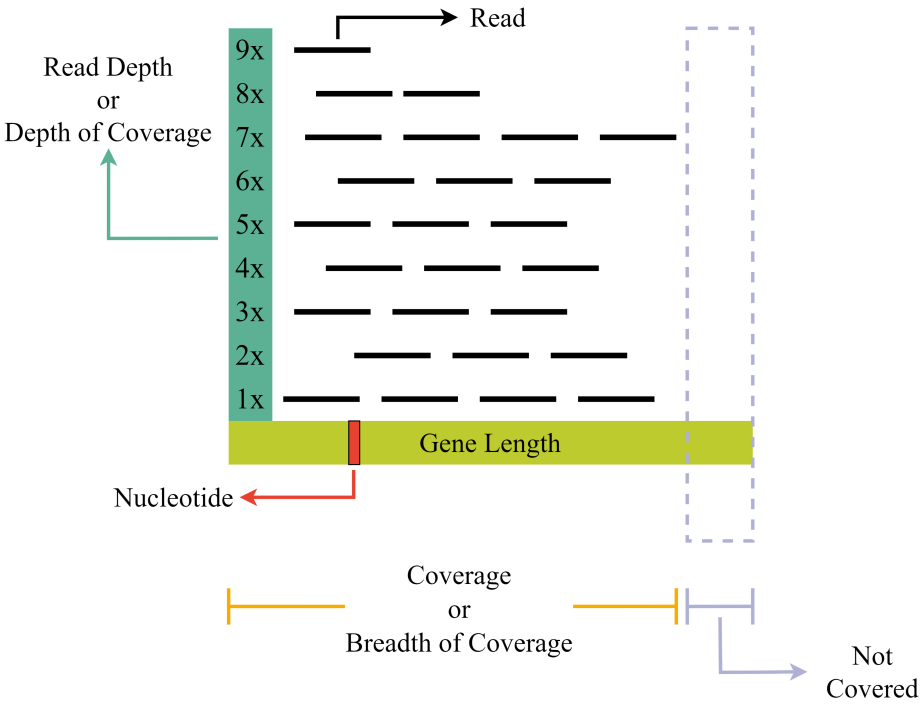



**Figure 1.8:** Scheme related to Coverage/ Breadth of Coverage and Read Depth/ Depth of Coverage in a gene. In this case, approximately 10% of the gene is depicted as not covered, and the depth reaches up to 9x. Adapted from [40]

Given the critical importance of validation in next-generation sequencing (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 average read depth and 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

*"I've learned a painful lesson, that for small programs dynamic typing is great. For large programs, you have to have a more disciplined approach. And it helps if the language actually gives you that discipline, rather than telling you, 'Well, you can do whatever you want.'" - Guido van Rossum, Python's creator*

Falar de Metodologia Agile

## 2.1 Requirements

This section presents the requirements for the developed software, categorized into functional and non-functional requirements.

### 2.1.1 Functional Requirements

Functional requirements 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. [41] The main functional requirements of the developed software are:

***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 Coverage/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 exon 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 CSV file.

### 2.1.2 Non-Functional Requirements

Non-functional requirements 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. [41] 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 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 functional and non-functional requirements, 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 they wish to perform. 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 uploads a BAM or CRAM file, containing the sequencing data. Additionally, they 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 read depth at each genomic position. This file is processed by a Python script that calculates two key metrics: 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 file.

***Gene Panel Analysis***

Similar to the Single Gene analysis, the user uploads a BAM/CRAM file and receives 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. 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 uploads the BAM/CRAM file, and the universal BED file corresponding to the exome is utilized. As in the other modes, SAMTOOLS generates the DEPTH file, which is processed to calculate the coverage metrics. The results are displayed or exported in the same manner.

The 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 coverage 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 (such as whole exome).

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

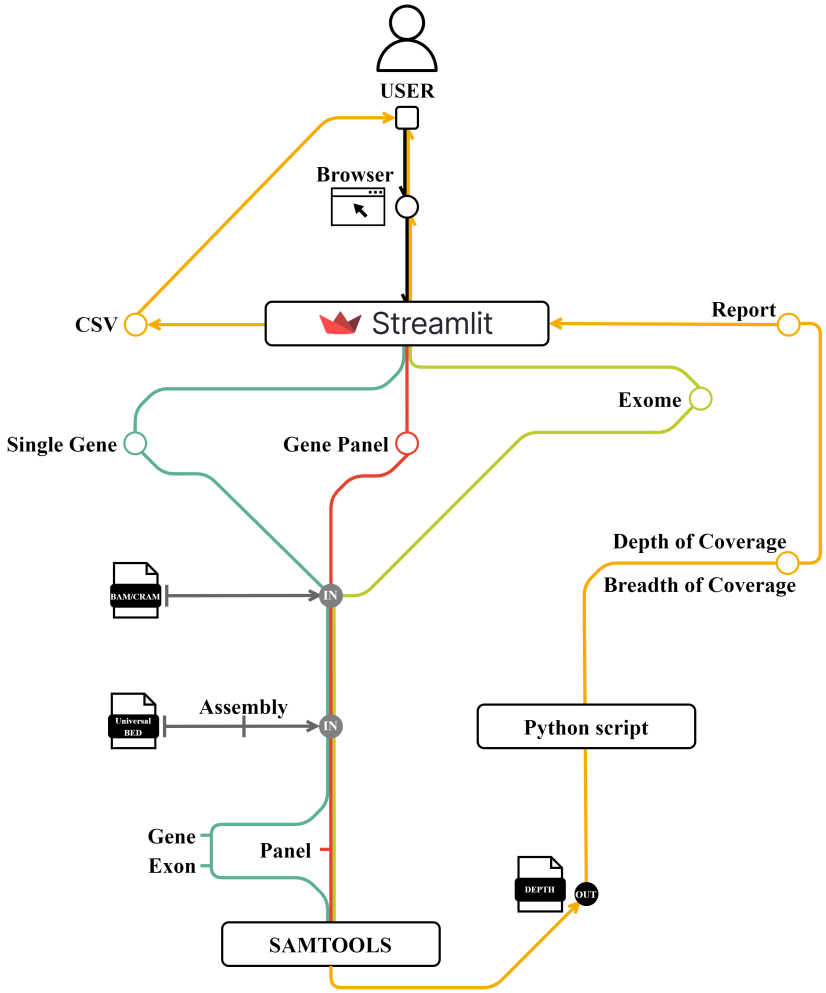



**Figure 2.1:** Scheme of the software architecture.

The project is organized in a modular structure to support the development of a metrics-based application. 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., exome, 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. The Figure 2.2 shows the software directory structure.

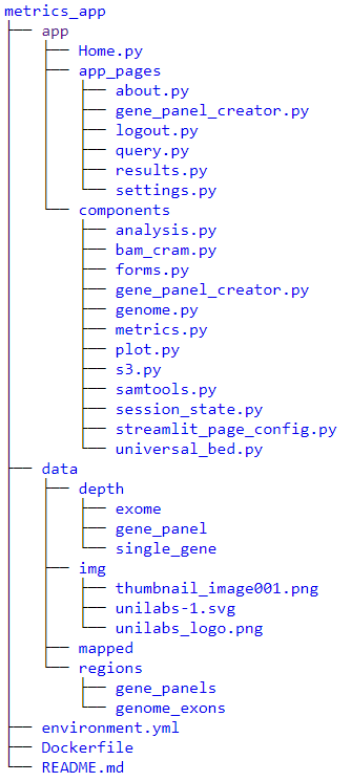



**Figure 2.2:** Software directory structure.

## 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 upload BAM or CRAM files and analyze specific genes and exons 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.

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.

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 (WSL)*

On Windows, developers have access to both the Windows and Linux environments, thanks to the Windows Subsystem for Linux (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. [42]

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, we followed the set of steps recommended on the Microsoft website to configure this environment (Build 19041 or higher). [42]

#### *Anaconda and Conda*

After installing WSL, the installation step of Anaconda followed, a platform for data science in Python/R that includes conda, a package and environment manager, making it easier for users to manage a collection of more than 7,500 open source packages. [43]

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. [44]

Following the documentation provided by Anaconda, the installation and creation of the conda environment was carried out. [45]

Additionally, all the dependencies of the Table A.4 were installed within the created environment. This installation was carried out by installing package by package, however, an environment.yaml file was made available that allows the bulk installation [46] 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. [47]

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. [47]

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. [47]

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. [47]

### 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. [48]

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. [48]

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. [49]

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 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 [50].

### 2.3.3 Processing and Simplification of BED File for Genomic Analysis

The original BED files utilized in this study was highly complex due to its rich annotation, particularly in the fourth column, which contained multiple genes and clinical identifiers associated with the same genomic region. For instance, a single entry could include a large number of comma-separated values representing multiple genes and clinical IDs. Such complexity was unnecessary for our specific analysis, which focused primarily on gene panel-level, gene-level and exon-level metrics. Therefore, it was essential to simplify the file while preserving relevant genomic data, particularly the coodinates, gene names and exon numbers.

The simplification process involved writing a custom Python script designed to extract and reorganize key elements from the original BED file. The script processes each line of the file, extracting the chromosome, start, and end positions, while simplifying the fourth column to retain only the first gene listed. Additionally, the script calculates exon numbers for each gene based on the sequence of occurrences in the file. This transformation results in a more concise representation, which is more suitable for downstream analysis.

The original file included regions defined by start and end positions of genomic intervals, as well as various annotations. A key challenge was to disambiguate the fourth column, where multiple genes and identifiers were grouped together. The Python script divided the contents of this column by the comma delimiter and selected only the first gene. In doing so, it reduced complexity while retaining a representative gene for each genomic region.

Another critical aspect of the transformation was the counting of exons for each gene. The script tracks changes in the gene name between consecutive lines and increments an exon count whenever a new gene is encountered. This ensures that each gene is assigned a unique exon number, allowing for consistent tracking of exon-level metrics. Additionally, the script calculates the length of each genomic interval by subtracting the start position from the end position.

To facilitate downstream analysis and ensure compatibility with different tools, the script also included an option to remove the "chr" prefix from chromosome identifiers. This was particularly important for certain bioinformatics tools that require the chromosome numbers without this prefix. The option to remove the prefix was controlled by a boolean flag, ensuring flexibility in handling different file formats.

The output of this process was a simplified BED file containing six columns: chromosome number, start position, end position, gene name, exon number, and exon length (calculated as the difference between the start and end positions). Below is an example of the resulting file structure:

```
(...)
17 41177976 41178064 RND2 3 88
17 41179199 41179309 RND2 4 110
17 41180077 41180212 RND2 5 135
17 41180448 41180697 RND2 6 249
17 41181106 41181131 RND2 7 25
17 41196309 41196310 SNORA40 1 1
17 41196331 41196332 BRCA1 1 1
17 41196371 41196372 BRCA1 2 1
17 41196402 41196403 BRCA1 3 1
(...)
```

In this example, the genes spans several lines, each representing a distinct exon. The

exon numbers were automatically assigned based on the gene's occurrence in the file, while the genomic intervals for each exon were derived directly from the original BED file. The final output, now simplified, facilitates a more straightforward analysis of coverage and depth metrics at both the gene and exon levels.

The Python script used for this transformation is outlined below:

```python
def process_bed_file(input_file, output_file, remove_chr_prefix=False):
    last_gene = None
    exon_count = 0

    with open(input_file, 'r') as infile, open(output_file, 'w') as outfile:
        for line in infile:
            fields = line.strip().split('\t')

            # Dividing column 4 by the ',' separator
            gene_info = fields[3].split(',')
            gene = gene_info[0]

            # Incrementing exon count when a new gene is encountered
            if gene != last_gene:
                exon_count = 1
            else:
                exon_count += 1

            # Calculating the difference between the end and start positions
            start = int(fields[1])
            end = int(fields[2])
            diff = end - start

            # Optionally removing the 'chr' prefix from chromosome number
            chromosome = fields[0]
            if remove_chr_prefix:
                chromosome = chromosome[3:] if chromosome.startswith("chr") else chromosome

            # Updating the last gene for the next iteration
            last_gene = gene

            # Writing the simplified output to the file
            outfile.write(f"{chromosome}\t{fields[1]}\t{fields[2]}\t{gene}\t{exon_count}\t
            ↪ {diff}\n")

# Example usage of the function with the option to remove 'chr' prefix
input_file = 'data/regions/genome_exons/hg19_Twist_ILMN_Exome_2.0_Plus_Panel_annotated.BED'
output_file_with_chr =
↪  'data/regions/genome_exons/hg19_Twist_ILMN_Exome_2.0_Plus_Panel_annotated_modif.bed'
output_file_no_chr = 'data/regions/genome_exons/hg19_Twist_ILMN_Exome_2.0_Plus_Panel_annot
↪  ated_modif_nochr.bed'

process_bed_file(input_file, output_file_with_chr, remove_chr_prefix=False)
```

```
process_bed_file(input_file, output_file_no_chr, remove_chr_prefix=True)
```

This script efficiently reduces the complexity of the original BED file, preparing it for further analysis and ensuring compatibility with tools that expect a simpler file format.

### 2.3.4 SAMtools

SAMtools is a fundamental tool for manipulating and analyzing DNA sequencing data, introduced in 2009. It supports a wide array of operations on SAM, BAM, and CRAM files, including format conversion, sorting, querying, indexing, and statistical analysis, with the "depth" function being central to coverage calculations in sequencing experiments [51].

The following section details a Python function that utilizes SAMtools and incorporates it into a Streamlit workflow for calculating the depth of coverage across specific genes and exons in sequencing files. The function allows users to filter the regions of interest and manage outputs dynamically through Streamlit's session state, offering flexibility for both interactive use and programmatic automation.

#### *Overview of the Depth Calculation Function*

The function **depth** calculates the depth of coverage for a set of specified genes and exons in a CRAM or BAM file using SAMtools and Streamlit session state. It provides mechanisms for filtering genomic regions through a BED file and storing intermediate results, such as filtered regions and depth outputs, within the session state for further use.

```
import subprocess
import streamlit as st
import os

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 1:** Initial setup for depth calculation.

### *Validation and Setup*

The first step involves validating the provided file paths for the CRAM/BAM file and the BED file. If any of these paths are incorrect, the function raises an error to prevent further execution with invalid data (Code 2). The function also converts gene and exon selections into appropriate formats if necessary.

The function handles both cases where a gene or exon selection is provided and where no such filtering is needed. If no filters are applied, the BED file is used as-is, while filtered regions are stored in Streamlit's session state for further processing.

```

    if not os.path.isfile(cram_path) or not os.path.isfile(bed_path):
        raise FileNotFoundError("Invalid file path.")

    # 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:
        # Load the original BED content directly
        with open(bed_path, 'r') as bed_file:
            st.session_state.filtered_bed = bed_file.read()  # Store original BED content
            ↪  in session state
```

**Code 2:** Handling gene and exon selection for BED file filtering.

### *Dynamic BED File Filtering*

When gene or exon selections are provided, the function reads and filters the BED file dynamically, matching entries to the specified genes and exons. The filtered results are then stored in the Streamlit session state (Code 3). This step allows flexible customization of regions of interest.

```
    else:
        # Otherwise, filter the BED file based on the gene and exon selection
        filtered_bed_lines = []
        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
```

```
11
12              chrom, start, end, gene, exon, size = columns[0], columns[1], columns[2],
                ↪  columns[3], columns[4], columns[5]
13
14              # Apply gene filter (multiple genes allowed)
15              if gene_selection is None or gene in gene_selection:
16                  # Apply exon filter: if exon_selection is None, include all exons for
                    ↪  the gene
17                  if exon_selection is None or exon in exon_selection:
18                      filtered_bed_lines.append(f"{chrom}\t{start}\t{end}\t{gene}\t{exon
                        ↪  }\t{size}\n")
19
20      # Check if any filtered BED content was found
21      if not filtered_bed_lines:
22          raise ValueError("No matching regions found for the provided gene or exon
            ↪  selection.")
23
24      # Store filtered BED content in Streamlit session state
25      st.session_state.filtered_bed = ''.join(filtered_bed_lines)
```

**Code 3:** Filtering BED file based on gene and exon selections.

### *Running SAMtools and Storing Output*

Once the filtered regions are stored, the function constructs a SAMtools command to calculate the depth of coverage. The filtered regions, stored in session state, are passed as input to SAMtools through standard input. The depth calculation is executed, and the results are either saved directly in Streamlit's session state or written to an output file in a specified directory (Code 4 and 5).

```
1   # Create a unique key for the output based on the CRAM/BAM file name
2   output_key = os.path.splitext(os.path.basename(cram_path))[0]
3
4   # Run samtools depth using the BED content (either original or filtered) from session
    ↪  state
5   samtools_command = ['samtools', 'depth', '-b', '-', cram_path]
6   samtools_process = subprocess.Popen(samtools_command, stdin=subprocess.PIPE,
    ↪  stdout=subprocess.PIPE, text=True)
7   samtools_output, _ = samtools_process.communicate(input=st.session_state.filtered_bed)
8
9   # Store samtools output (.depth content) in Streamlit session state as a dictionary
10  if 'depth_output' not in st.session_state:
11      st.session_state.depth_output = {}
12
13  st.session_state.depth_output[output_key] = samtools_output
```

**Code 4:** Running SAMtools and storing depth output in session state.

```
    if depth_dir:
        # 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)
```

**Code 5:** Saving the SAMtools depth output to a file.

***Example Usage***

This example demonstrates how to use the **depth** function to calculate the depth of coverage for the gene **BRCA1** across exons 1, 2, and 3 from a CRAM file. The results are stored in the specified output directory.

```
    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 6:** Example usage of the depth function.

### 2.3.5 Python script for metrics calculation

The following section explains a Python script designed to calculate sequencing coverage metrics based on depth of coverage data from BAM/CRAM files and gene/exon information from a BED file. This function operates within a Streamlit session, dynamically filtering data and calculating coverage statistics for different genes and exons. The results are stored and presented in a structured format, making it highly flexible for genomic data analysis workflows.

***Initialization of Metrics***

The function begins by defining a reusable structure for storing the various metrics that will be calculated during the process. The function `initialize_metrics()` creates a dictionary containing keys for metrics such as "Size Coding", "Size Covered", "Average Read Depth", and several thresholds for coverage percentage, such as "Coverage % (1x)" and "Coverage % (500x)" (Code 7). This structure will be used repeatedly to initialize metrics for both genes and exons.

```
def initialize_metrics():
    return {
        'Size Coding': 0,
        'Size Covered': 0,
        'Average Read Depth': 0,
        'Min Read Depth': 0,
```

```
        'Max Read Depth': 0,
        'Coverage % (1x)': 0,
        'Coverage % (10x)': 0,
        'Coverage % (15x)': 0,
        'Coverage % (20x)': 0,
        'Coverage % (30x)': 0,
        'Coverage % (50x)': 0,
        'Coverage % (100x)': 0,
        'Coverage % (500x)': 0,
        'Coverage (>500x)': 0,
        'Coverage (0-1x)': 0,
        'Coverage (2-10x)': 0,
        'Coverage (11-15x)': 0,
        'Coverage (16-20x)': 0,
        'Coverage (21-30x)': 0,
        'Coverage (31-50x)': 0,
        'Coverage (51-100x)': 0,
        'Coverage (101-500x)': 0
    }
```

**Code 7:** Initialization of metrics structure.

### *Accessing Filtered Data from Session State*

The next step is to define the `calculate_metrics()` function, which retrieves filtered BED content and depth data from the Streamlit session state (Code 8). These datasets represent the regions of interest (filtered gene/exon regions) and the corresponding depth of coverage information. The function first checks if these datasets exist in session state. If not, an error is raised to indicate missing data.

```
def calculate_metrics():
    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.")
```

**Code 8:** Accessing filtered BED and depth data from session state.

### *Reading the Filtered BED File*

The filtered BED content, representing the genomic regions of interest, is then read into a pandas DataFrame (Code 9). The BED file contains columns such as chromosome (`CHROM`), start and end positions (`START`, `END`), gene names (`GENE`), exon numbers (`EXON`), and exon sizes (`SIZE`). This DataFrame will be used later to match the regions in the depth file and calculate the metrics.

```
bed_df = pd.read_csv(io.StringIO(bed_content), sep='\t', header=None, names=['CHROM', 'START', 'END', 'GENE', 'EXON', 'SIZE'])
```

**Code 9:** Reading filtered BED content into a DataFrame.

### *Processing the Depth Data*

For each CRAM/BAM file, the corresponding depth data is also read into a pandas DataFrame. The depth data contains three columns: chromosome (`CHROM`), position (`POS`), and depth (`DEPTH`) at each position (Code 10). These values will be used to calculate various metrics for each gene and exon.

```
for file_name, depth_content in depth_dict.items():
    depth_df = pd.read_csv(io.StringIO(depth_content), sep='\t', header=None, names=['CHROM', 'POS', 'DEPTH'])
```

**Code 10:** Reading depth data for each CRAM/BAM file.

### *Calculating Metrics for All Genes*

Once the data is loaded, the script begins calculating metrics for all the genes combined. For example, it computes the total "Size Coding" by summing the sizes of all exons from the BED file, and the "Average Read Depth" by averaging the depths from the depth file. To calculate the "Average Read Depth (Gene Weighted)", a weighted sum of depths is computed based on the size of each gene (Code 11).

```
all_genes_metrics['Size Coding'] = bed_df['SIZE'].sum()
all_genes_metrics['Average Read Depth'] = all_depths.mean()

weighted_sum = 0
total_size = 0
for _, row in bed_df.iterrows():
    start = row['START']
    end = row['END']
    size = row['SIZE']
    gene_depths = depth_df[(depth_df['POS'] >= start) & (depth_df['POS'] <= end)]['DEPTH']

    weighted_sum += gene_depths.sum() * size
    total_size += size

if total_size > 0:
    all_genes_metrics['Average Read Depth (Gene Weighted)'] = weighted_sum / total_size
else:
    all_genes_metrics['Average Read Depth (Gene Weighted)'] = 0
```

**Code 11:** Calculating gene metrics, including weighted average read depth.

### *Calculating Coverage Percentages*

The function also calculates coverage percentages for various depth thresholds, such as 1x, 10x, 15x, etc. These percentages are computed by counting the number of positions in the depth data where the coverage is greater than or equal to each threshold and then dividing by the total number of positions (Code 12). Additionally, depth intervals such as "Coverage (0-1x)" and "Coverage (101-500x)" are calculated.

```
    coverage_thresholds = [1, 10, 15, 20, 30, 50, 100, 500]
    coverage_counts = {threshold: (all_depths >= threshold).sum() for threshold in
    ↪  coverage_thresholds}

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

**Code 12:** Calculating coverage percentages for different thresholds.

### *Per-Gene and Per-Exon Metrics*

For each gene, the function calculates individual metrics by iterating over the filtered BED regions and depth data specific to that gene. Similarly, metrics for each exon are computed by breaking down the depth data for each exon. This per-gene and per-exon calculation mirrors the process used for all genes but is applied to specific regions (Code 13 and 14).

```
    for gene in genes:
        gene_metrics = initialize_metrics()
        gene_bed_df = bed_df[bed_df['GENE'] == gene]
        gene_depths = pd.Series(dtype=float)

        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)
```

**Code 13:** Calculating metrics for each gene.

```
    for exon_name in gene_bed_df['EXON'].unique():
        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)
```

**Code 14:** Calculating metrics for each exon.

***Final Output Structure***

The final results are stored in a hierarchical structure. For each CRAM/BAM file, the function stores overall metrics for all genes, per-gene metrics, and per-exon metrics. These results are returned as a dictionary, which can be further processed or displayed in the Streamlit application (Code 15).

```
    results[file_name]['All Genes'] = all_genes_metrics
    results[file_name]['Genes'] = genes_data
    results[file_name]['Exons'] = exons_data

    return results
```

**Code 15:** Storing and returning the calculated metrics.

### 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 coverage data, organizes the results into tables, and provides functionality to download a PDF report. It dynamically handles multiple samples, allowing users to interactively select and filter metrics.

***Initializing the Application and Displaying Logos***

The script begins by importing necessary libraries, including Streamlit for the web interface, pandas for data manipulation, and WeasyPrint for generating PDF reports. Two logo images are displayed at the beginning of the app: one for the sidebar and another in the main body (Code 16). Streamlit's `st.logo()` method is used to load and display these logos.

```
import streamlit as st
import pandas as pd
from components import metrics
import numpy as np
import io
from weasyprint import HTML
import base64
import datetime

sidebar_logo = "data/img/unilabs_logo.png"
main_body_logo = "data/img/thumbnail_image001.png"
st.logo(sidebar_logo)
st.logo(main_body_logo)

st.title("Results")
```

**Code 16:** Initializing the Streamlit app and displaying logos.

### *Defining the Desired Metrics Order*

The next step is to define the desired order for displaying metrics. This order ensures consistency across different DataFrames for genes and exons, listing metrics like "Size Coding", "Average Read Depth", and several coverage thresholds (Code 17).

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

**Code 17:** Defining the desired metrics order.

### *Calling the Metrics Calculation Function*

The script calls the `calculate_metrics()` function, defined in the `metrics` module, to compute sequencing coverage metrics for one or multiple samples. The results are stored in a dictionary, with the sample file names as keys (Code 18). Each file contains metrics for one or multiple genes individually and together, and exons.

```
results = metrics.calculate_metrics()
file_names = list(results.keys())
```

**Code 18:** Calling the `calculate_metrics()` function to retrieve results.

### *Preparing the All Genes DataFrame*

To display metrics for all genes, the script constructs a DataFrame. The column `Metric` lists the metric names from `desired_order`, and each subsequent column corresponds to a sample file. For each sample, the metrics values are extracted from the results dictionary (Code 19). This table allows users to view metrics for all genes combined across multiple samples.

```
all_metrics = desired_order
all_metrics.insert(3, 'Average Read Depth (Gene Weighted)')
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 19:** Constructing the DataFrame for all genes metrics.

### *Preparing the Genes DataFrames*

The script creates individual DataFrames for each gene by iterating over the results for all samples. First, a set of all genes is compiled. For each gene, the corresponding metrics are extracted from each sample and stored in a DataFrame. The resulting DataFrames are stored in a dictionary for easy access (Code 20).

```
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 20:** Preparing individual DataFrames for each gene.

### *Preparing the Exons DataFrames*

The script handles exon-level metrics similarly to gene-level metrics. For each gene, a set of exons is compiled, and DataFrames are created for each exon across all samples. These DataFrames are stored in a nested dictionary, where the outer key is the gene name and the inner key is the exon name (Code 21).

```
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 21:** Preparing DataFrames for exon-level metrics.

### *Downloading the Report*

In the report generation section, users are provided with an option to select a sample and download a PDF report. The current date and time are included in the report, and the metrics for the selected sample are added to the HTML content. WeasyPrint is used to convert the HTML string into a PDF file, which users can download (Code 22).

```
with st.status("Building the report...")as status:
    if len(file_names) > 1:
        selected_sample = st.selectbox("Select Sample", file_names)
    else:
        selected_sample = file_names[0]

    report_date = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    html_content = """
    <html>
    <head>
    <style>...</style>
    </head>
    <body>
    """

    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>'
    html_content += f'<p>Report generated on: {report_date}</p>'
    html_content += f'<h2>Overview - Sample: {selected_sample}</h2>'

    overview_df = all_genes_df[['Metric', selected_sample]]
    html_content += overview_df.to_html(index=False)

    html_content += '</body></html>'
    html_obj = HTML(string=html_content)
    pdf_bytes = html_obj.write_pdf()

    st.download_button(
        label="Download",
        data=pdf_bytes,
        file_name=f'report_{selected_sample}.pdf',
        mime='application/pdf'
    )
```

**Code 22:** Generating and downloading a PDF report.

### *Displaying Tabs for Metrics*

The script determines which tabs to display based on the type of analysis (gene panel, single gene, or exome). It dynamically creates tabs for "Overview", "Gene Detail", and "Exon

Detail" depending on the analysis type. Within each tab, users can filter and display metrics using checkboxes (Code 23 and 24).

```
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 = []

tabs = st.tabs(tab_names)
tab_dict = dict(zip(tab_names, tabs))
```

**Code 23:** Creating tabs for gene and exon metrics.

Each tab contains a popover filter for selecting metrics to display. Users can select or deselect all metrics using a button, and the displayed DataFrame is updated based on the selected metrics (Code 24).

```
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

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

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

        with st.popover("Filters"):
            st.subheader("Select Metrics to Display")

            all_selected = all(
                st.session_state.get(f"tab1_metric_{metric}", False)
                for metrics_list in metrics_dict.values()
                for metric in metrics_list
            )
```

```
26
27              if st.button("Select All" if not all_selected else "Deselect All",
                ↪  key="tab1_select_all"):
28                  select_all_metrics(not all_selected, metrics_dict, "tab1")
29                  st.experimental_rerun()
30
31              col1, col2, col3 = st.columns(3)
32              metrics_keys = list(metrics_dict.keys())
33
34              for i, section in enumerate(metrics_keys):
35                  with [col1, col2, col3][i % 3]:
36                      st.write(f"**{section}**")
37                      for metric in metrics_dict[section]:
38                          st.checkbox(metric, key=f"tab1_metric_{metric}")
39
40          selected_metrics = [
41              metric
42              for metrics_list in metrics_dict.values()
43              for metric in metrics_list
44              if st.session_state.get(f"tab1_metric_{metric}", False)
45          ]
46
47          metrics_df = all_genes_df[all_genes_df['Metric'].isin(selected_metrics)].reset_ind
            ↪  ex(drop=True)
48          st.dataframe(metrics_df, hide_index=True, height=738, width=800)
```

**Code 24:** Displaying metrics in the "Overview" tab with filters.

### *Gene Detail and Exon Detail Tabs*

The "Gene Detail" and "Exon Detail" tabs allow users to select a gene or exon and view the corresponding metrics. The same popover filter system used in the "Overview" tab is applied here to allow filtering of metrics. The script iterates through the list of genes or exons, and users can select the desired gene or exon to view detailed metrics (Code 25 and 26).

```
1   if "Gene Detail" in tab_dict:
2       with tab_dict["Gene Detail"]:
3           st.write(f"Date: {report_date}")
4           st.write(f"Analyzing file(s): {file_names}")
5
6           with st.container():
7               if not genes_list:
8                   st.warning("No genes found in the results.")
9               else:
10                  gene = st.selectbox("Select Gene", genes_list, key="gene_selectbox")
11                  gene_metrics_df = genes_dfs[gene]
12
13                  metrics_dict = {
14                      "Basic Information": ["Average Read Depth", "Size Coding", "Size
                        ↪  Covered"],
```

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

    with st.popover("Filters"):
        st.subheader("Select Metrics to Display")

        all_selected = all(
            st.session_state.get(f"tab2_metric_{metric}", False)
            for metrics_list in metrics_dict.values()
            for metric in metrics_list
        )

        if st.button("Select All" if not all_selected else "Deselect All",
        ↪ key="tab2_select_all"):
            select_all_metrics(not all_selected, metrics_dict, "tab2")
            st.experimental_rerun()

        col1, col2, col3 = st.columns(3)
        metrics_keys = list(metrics_dict.keys())

        for i, section in enumerate(metrics_keys):
            with [col1, col2, col3][i % 3]:
                st.write(f"**{section}**")
                for metric in metrics_dict[section]:
                    st.checkbox(metric, key=f"tab2_metric_{metric}")

    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']
    st.dataframe(df[final_metrics], hide_index=True, height=738, width=800)
```

**Code 25:** Displaying metrics in the "Gene Detail" tab with filters.

```
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]

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

                    with st.popover("Filters"):
                        st.subheader("Select Metrics to Display")

                        all_selected = all(
                            st.session_state.get(f"tab3_metric_{metric}", False)
                            for metrics_list in metrics_dict.values()
                            for metric in metrics_list
                        )

                        if st.button("Select All" if not all_selected else "Deselect All", key="tab3_select_all"):
                            select_all_metrics(not all_selected, metrics_dict, "tab3")
                            st.experimental_rerun()

                        col1, col2, col3 = st.columns(3)
```

```
                        metrics_keys = list(metrics_dict.keys())

                        for i, section in enumerate(metrics_keys):
                            with [col1, col2, col3][i % 3]:
                                st.write(f"**{section}**")
                                for metric in metrics_dict[section]:
                                    st.checkbox(metric, key=f"tab3_metric_{metric}")

                    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']
                    st.dataframe(df[final_metrics], hide_index=True, height=738, width=800)
```

**Code 26:** Displaying metrics in the "Exon Detail" tab with filters.

## 2.4 Deployment

CHAPTER 3

# Results

*"The only source of knowledge is experience." - Albert Einstein*

## 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 coverage metrics 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 the results table below. 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.

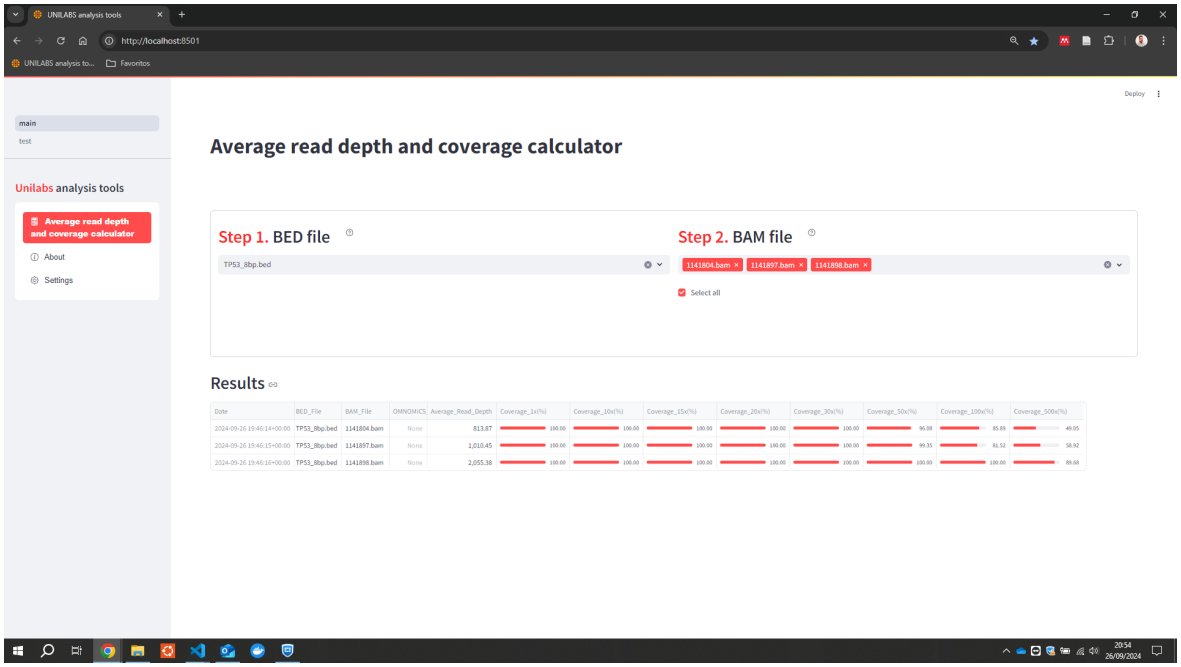



**Figure 3.1:** First version of the GUI.

### 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.

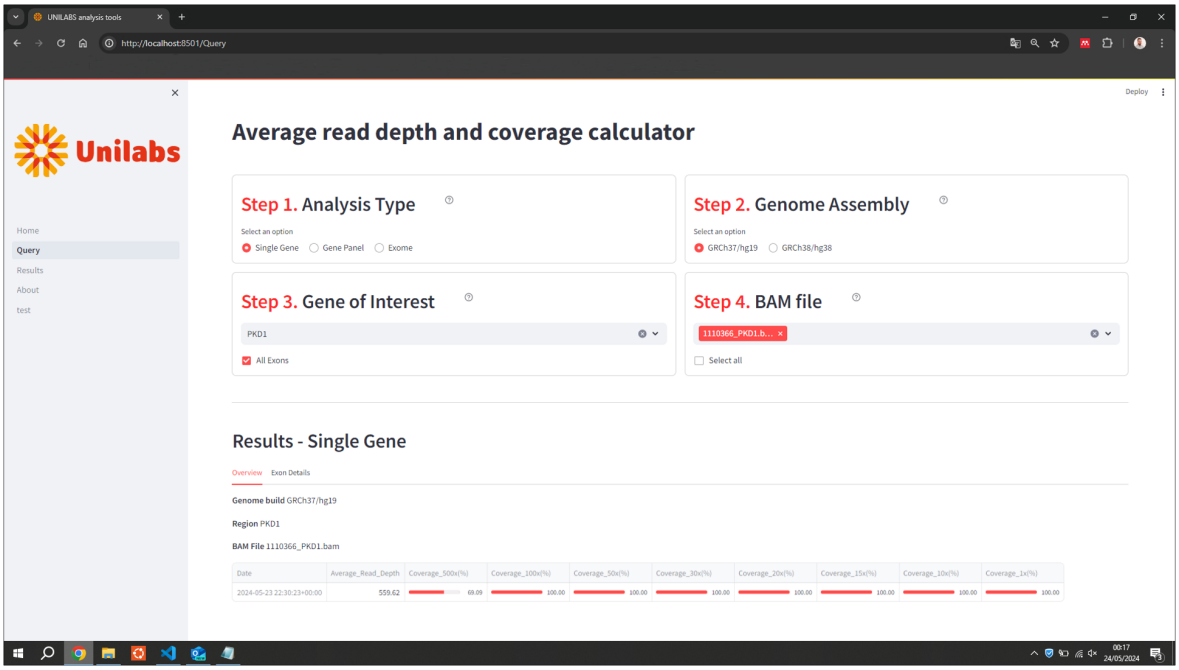



**Figure 3.2:** Second version of the GUI.

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, making it a powerful tool for researchers.

### 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 coverage metrics 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 read depth and 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. Users must input their credentials, and upon successful authentication, they are granted access to the full range of analysis tools.

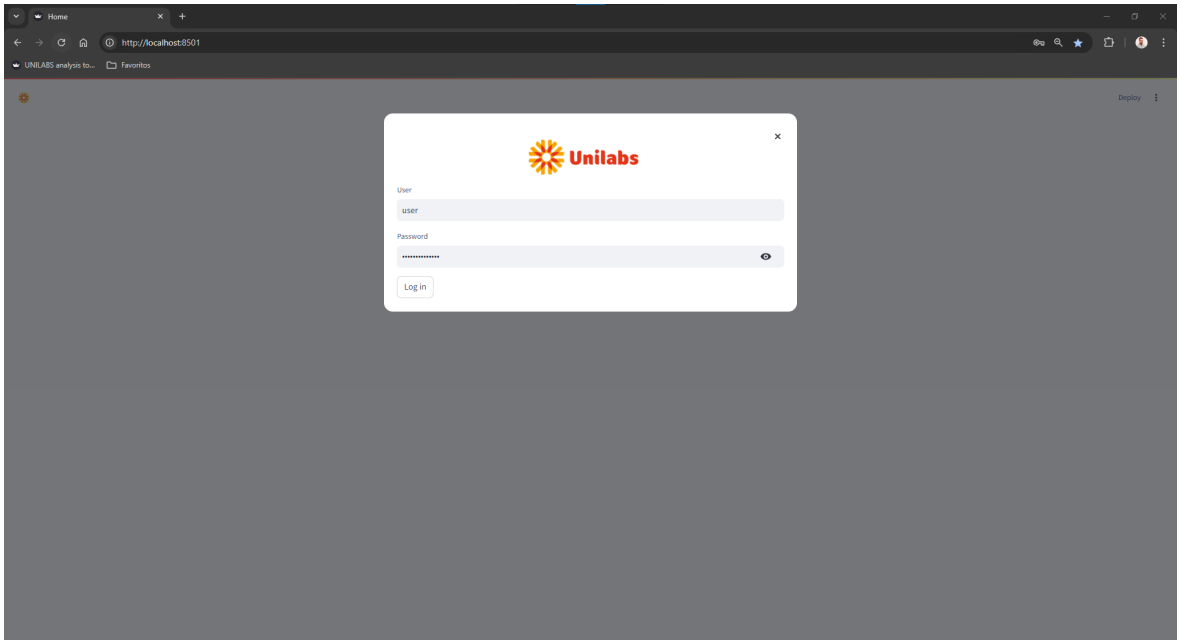



**Figure 3.3:** Login Interface for the Software

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.

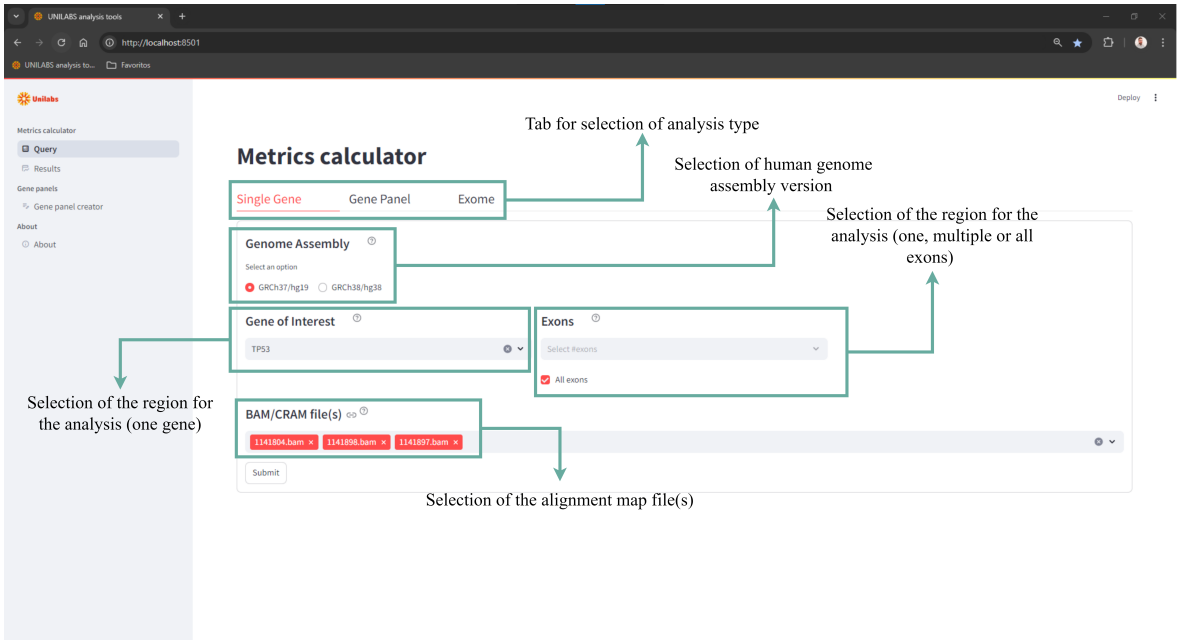



**Figure 3.4:** Single Gene Analysis Workflow

- **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 coverage percentages across different thresholds (e.g., 10x, 20x, 30x). This data is also available for download in a PDF format, ensuring users can retain a permanent copy of the analysis results.

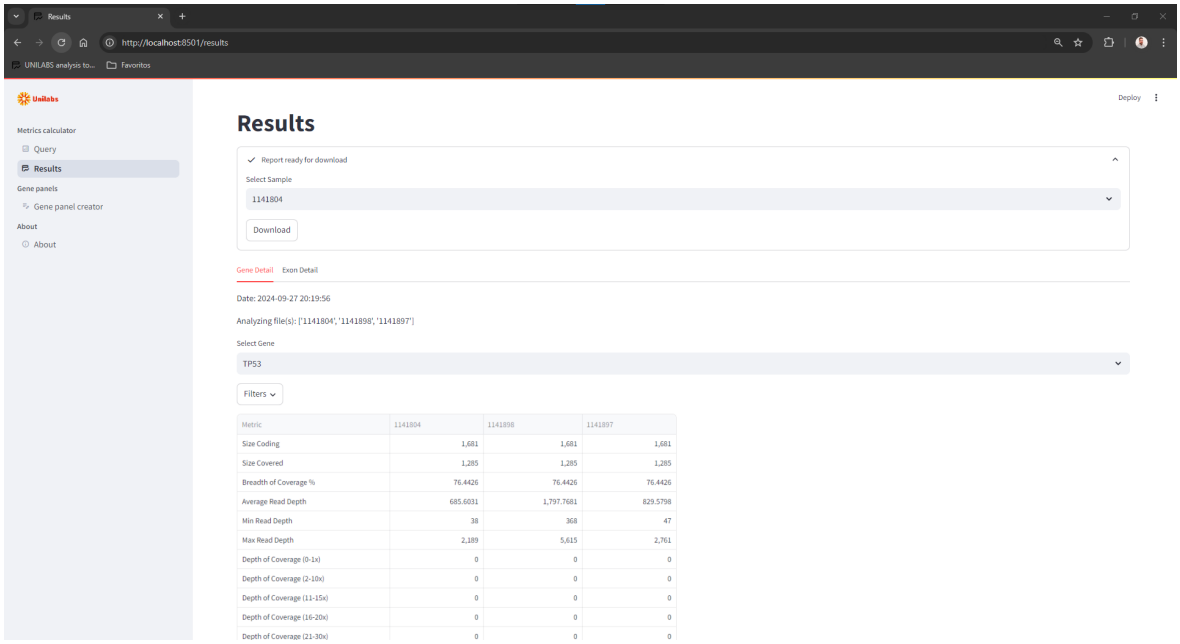



**Figure 3.5:** Results Tab Loading the Final Report

In Figure 3.5 and 3.6, the detailed metrics for the gene *TP53* are displayed, offering 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 *TP53* gene is 1,681 base pairs (bp), with a covered size of 1,285 bp for all three samples, resulting in a Breadth of Coverage of 76.4%. The average read depth across the three samples was 685.6x, 1,797.8x, and 829.6x, respectively. Sequencing depth, 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. [22]

When examining the depth of coverage across different thresholds, the coverage percentages for the 10x, 20x, and 30x thresholds were consistently 100% for all three 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 95.3%, 100%, and 99.2%, respectively. Similarly, at the 100x threshold, the coverage percentages were 83.8%, 100%, and 78.5%. Finally, at the highest threshold of 500x, the coverage percentages dropped more significantly, with values of 41.4%, 88.2%, and 52.8%, respectively.

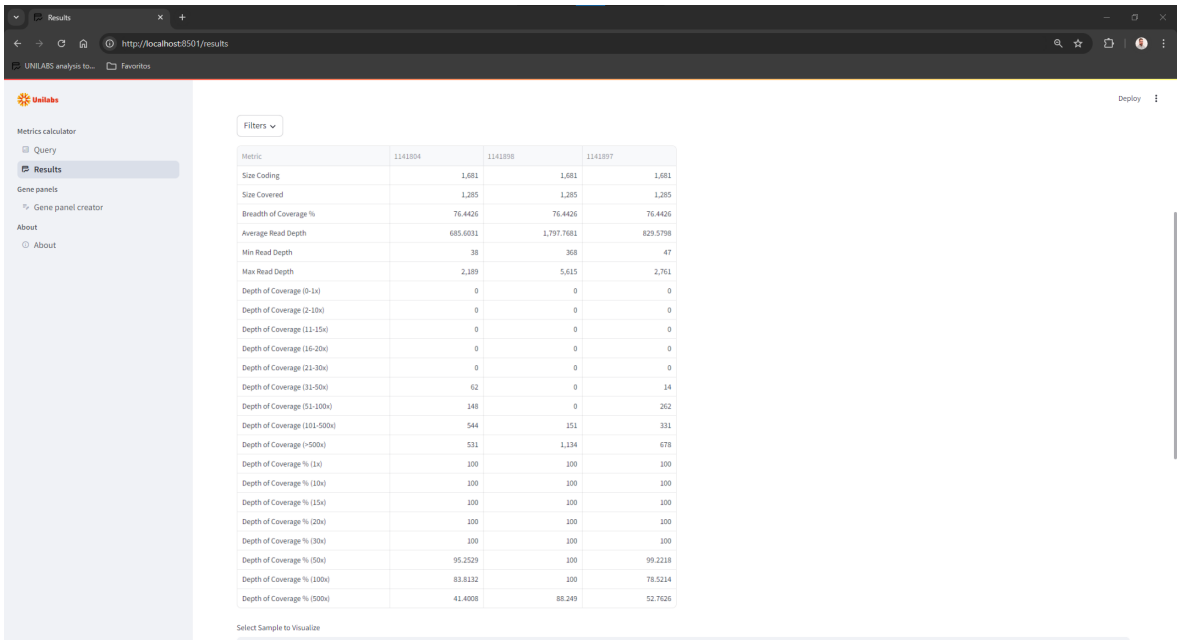



**Figure 3.6:** Final Report with Detailed Metrics for Gene TP53

- **Depth of Coverage Visualization**

  One of the critical features of the final software version is its ability to visualize depth

of coverage, 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.

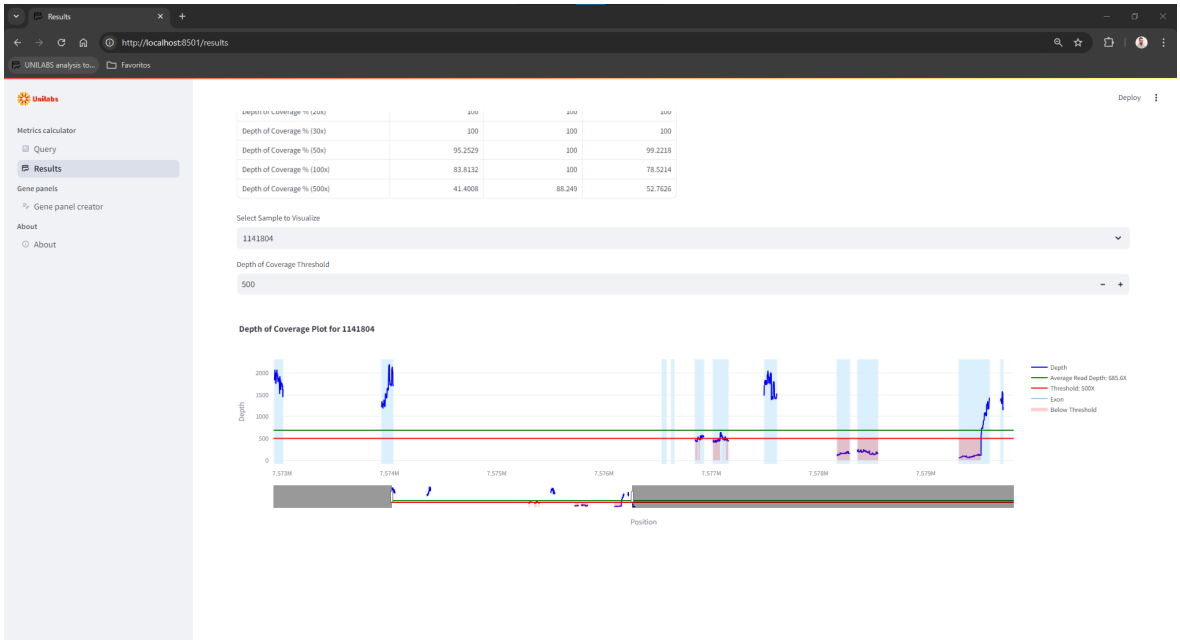



**Figure 3.7:** Depth of Coverage Visualization for Gene TP53

### *Gene Panel Analysis*

## 3.2 Test and validation

De forma a validar a ferramenta, foram realizados testes com dados reais de sequenciação genómica. Os resultados obtidos foram comparados com os obtidos por outras ferramentas de análise de dados genómicos comerciais, como a *Omnomics*. Foram reproduzidas as análises para Singles Gene e Gene Panel. No primeiro caso, para o mesmo ficheiro de alinhamento .bam, e usando como referência a mesma versaão do genoma humano (hg19), os resultados obtidos foram semelhantes como se pode observar na Figura

## 3.3 Performance

## 3.4 Comparison with other tools

## 3.5 Users feedback

CHAPTER 4

# Additional activities during the internship

*"The only source of knowledge is experience." - Albert Einstein*

CHAPTER 5

# Discussion

*"The only source of knowledge is experience." - Albert Einstein*

## 5.1 SWOT Analysis

During the development of the genomic analysis software, a Strengths Weeknesses Opportunities and Threats (SWOT) analysis was performed to assess its strategic positioning. The analysis identified internal strengths, weaknesses, and external opportunities and threats within bioinformatics and genomics. Figure **??** summarize the key findings, outlining crucial elements that guided the software's development and deployment.

**Table 5.1**

**Strengths**

**1.** Advanced bioinformatics and genomics concepts were applied in the development of an independent software for essential genomic analysis metrics, such as vertical and horizontal sequencing depth, ensuring compliance with NGS guidelines.
**2.** The use of advanced technologies like WSL, Anaconda, Conda, Git, GitHub, Streamlit, and Python ensures robust and efficient development. The software features a user-friendly interface, making it accessible to all users, including those without bioinformatics expertise.
**3.** Git and GitHub enable code sharing, ensuring traceability and reproducibility, while Anaconda and Conda create an isolated and reproducible environment for managing packages and dependencies.

**Weaknesses**

**1.** The requirement for WSL and a Linux environment may limit accessibility for users unfamiliar with these technologies.
**2.** While the graphical interface is intuitive, it may not suffice for advanced users who prefer command-line tools for analysis.
**3.** The initial setup of the development environment involves multiple steps and requires a solid understanding of the tools, which can be a barrier for less experienced users.
**4.** Comparing and validating metrics obtained by the new software with other tools is essential, but ensuring consistent and comparable results can be challenging.

**Oportunities**

**1.** The software can be expanded to include other important genomic analysis metrics.
**2.** Unilabs could promote internal use, establishing it as the standard tool for metric analysis.
**3.** Releasing the software as an open-source solution could attract a community of developers and users to contribute to its continuous improvement.
**4.** The rapid evolution of sequencing technologies and data analysis offers opportunities to continuously enhance and update the tool with new advancements and algorithms.

**Threats**

**1.** Several established tools and platforms in the market offer similar functionalities, which may hinder the adoption of the new software.
**2.** Competing with robust commercial software that offers extensive support can be challenging.
**3.** The rapid evolution of sequencing technologies and data analysis could render some features obsolete or require frequent updates.
**4.** Dependence on third-party software and libraries that may discontinue or significantly change their APIs presents a risk.
**5.** Increasing regulation around genomic data and genetic testing may impose additional challenges for the use and distribution of the software.

## 5.2 Optimization

## 5.3 Impact on the company

CHAPTER 6

# Final remarks

*"The only source of knowledge is experience." - Albert Einstein*

# Bibliography


[1] P. Bourque, R. E. Fairley, and I. C. Society, *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*, 3rd. Washington, DC, USA: IEEE Computer Society Press, 2014, ISBN: 0769551661.

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

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

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

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

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

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

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

[13] 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..

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

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

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

[24] 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.

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

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

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 3billion, *Sequencing depth vs coverage*, Aug. 2023. [Online]. Available: https://3billion.io/blog/sequencing-depth-vs-coverage.

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

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

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

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

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

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

[46] 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.

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

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

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

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

[51] P. Danecek, J. K. Bonfield, J. Liddle, *et al.*, «Twelve years of samtools and bcftools», *GigaScience*, vol. 10, 2 Feb. 2021, ISSN: 2047217X. DOI: 10.1093/gigascience/giab008.

[52] 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 [52]

| 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:** Samtools - BED file documentation

| Column | BED Field | Type | Regex or range | Brief description |
|---|---|---|---|---|
| 1 | chrom | String | [[a-zA-Z0-9_]1,255] | Chromosome name |
| 2 | chromStart | Int | [0, $2^{64}$ - 1] | Feature start position |
| 3 | chromEnd | Int | [0, $2^{64}$ - 1] | Feature end position |
| 4 | name | String | [\x20-\x7e]1,255 | Feature description |
| 5 | score | Int | [0, 1000] | A numerical value |
| 6 | strand | String | [-+.] | Feature strand |
| 7 | thickStart | Int | [0, $2^{64}$ - 1] | Thick start position |
| 8 | thickEnd | Int | [0, $2^{64}$ - 1] | Thick end position |
| 9 | itemRgb | Int,Int,Int | ([0, 255], [0, 255], [0, 255]) \| 0 | Display color |
| 10 | blockCount | Int | [0, chromEnd - chromStart] | Number of blocks |
| 11 | blockSizes | List[Int] | ([[0-9]]+,)blockCount-1[[0-9]]+,? | Block sizes |
| 12 | blockStarts | List[Int] | ([[0-9]]+,)blockCount-1[[0-9]]+,? | Block start positions |

**Table A.4:** Packages used in the project

# Package Name Version