TÉCNICO LISBOA

# Implementing Bioinformatics Pipelines and User Interfaces for Selection of Immunotherapeutic Targets in Colorectal Cancer

**António Manuel Farinha Gabriel Correia do Paulo**

Thesis to obtain the Master of Science Degree in

**Information Systems and Computer Engineering**

Supervisors: Dr. João André Nogueira Custódio Carriço
Dr. Dina Ruano

**Examination Committee**

Chairperson: Prof. Luís Manuel Antunes Veiga
Supervisor: Dr. João André Nogueira Custódio Carriço
Member of the Committee: Prof. Susana de Almeida Mendes Vinga Martins

**January 2021**

# Acknowledgments

I would like to start by thanking my mother for giving me all I have today. Without your care, worry and guidance I know I would not be where I am today. I was born with a special advantage, and that was given to me by you mom. On the same note, thank you Ana. When I was little you were my personal teacher (a very demanding one!). You also gave me a head start in life and with you I can always spend some time talking about whatever it is that is on my mind. You are always ready to help me and that is invaluable. Talking about always being there to help, thank you João for always being there to solve the unsolvable and to support me. I've always admired your outside the box thinking and felt the care you had for me.

Thank you Dina for the availability you showed to teach me and to help me, even outside LUMC. Remember those pesky Dutch bedbugs? I sure do! Your passion for bioinformatics is admirable. It was that passion that caught my attention in your presentation at Bioinformatics Open Days in Minho. The way you talked and the clearness with which you explained concepts that were totally new to me is something I will never forget. Thank you for the opportunity, for being my teacher and for becoming my friend.

Thanks João Carriço for giving me my first opportunity to get into the Bioinformatics world and for accepting to accompany me in this thesis adventure! I always enjoyed our meetups at iMM where you added a bit more knowledge to my compendium. You were possibly the first piece of the puzzle in this bioinformatics journey.

Thank you Noel for accepting me into the Immunogenomics group for a semester and for the warm welcome you gave me when I arrived to the group. You showed me that above being the group leader you were a kind person that would treat me as equal.

I would like to acknowledge the SASC team at LUMC for helping me in technical aspects and leave a special thanks to Davy Cats. You were always available to answer my questions, even if at times they were a bit more on the "philosophical" side of things. You made things considerably smoother, thanks!

A warm and very fuzzy thank you to Sofia and Nuno for being a big part of my life. You two taught me a lot throughout the years, namely when it comes to being a better person. Also thank you Martim for being just the way you are, a friend that truly cares.

Thank you Sílvia and Cruz. To this date I still feel very lucky to have come across you two in my academic life. I get quite nostalgic when I remember taking the Metro back home with you Sílvia. Keep on smiling the way only you know how to do. Cruz, thank you for teaching me to find and understand things on my own. I use that lesson everyday. I will never forget trying not to burst in laughter with you at the most inadequate of times.

Also a big thanks to all the anonymous people that created the support material I had access to all these years. Learning makes more sense when knowledge isn't hidden away. Thanks to Instituto Superior Técnico for providing invaluable opportunities for their students. I owe a lot to that.

And of course, the love of my life, Filipa. You are always there for me and helped me more than I could ever ask for. I love your creative thinking and the ability you have to come up with ideas like it is second nature. With you, I will never stop learning.

# Abstract

With the advent of Next-Generation Sequencing (NGS), tumor tissue can be characterized much faster and at a lower cost than before, opening the way for personalized treatments such as cancer vaccines. A key aspect in creating these vaccines is identifying the tumor-specific mutations of a given patient, known as neoantigens. Identifying neoantigens from patient samples involves processing a great amount of NGS data efficiently so that therapies can be developed timely. In this work, we present a neoantigen identification bioinformatics pipeline that is capable of leveraging High-Performance Computing clusters. The pipeline was developed with the purpose of analyzing low-mutation colorectal cancer but is not restricted to this type of cancer. Contrary to other pipelines, ours is complete in the sense that it covers all the steps required to turn sequenced tissue samples into ranked neoantigen predictions. Moreover, it makes use of RNA data to confirm that the tumor-specific mutations are being expressed. The pipeline is open-source, freely available and was designed with reproducibility, modularity and collaboration in mind. Additionally, this work includes a data visualization module through which users can visualize the pipelines' results. The pipeline was developed as an overhaul to a previously used pipeline by the Immunogenomics group at the Leiden University Medical Center, managing to reduce the execution time from days to hours, while keeping neoantigen prediction in line with previous results.

# Keywords

Neoantigen identification; Bioinformatics pipeline; Personalized cancer vaccine; Next-generation sequencing; High-Performance Computing

# Resumo

Com o advento da sequenciação de nova geração (NGS), é possível caracterizar as amostras de tecido tumoral mais rapidamente e a um custo inferior, abrindo o caminho para tratamentos personalizados como as vacinas contra o cancro. Um aspecto chave na criação destas vacinas é a identificação de mutações específicas nos tumores dos pacientes, conhecidas como neoantígenos. A identificação de neoantígenos a partir de amostras de pacientes envolve o processamento eficiente de uma grande quantidade de dados NGS, de modo a que as terapêuticas possam ser desenvolvidas atempadamente. Neste trabalho, apresentamos um pipeline bioinformático de identificação de neoantígenos capaz de utilizar clusters de computação de alto desempenho. O pipeline foi desenvolvido para analisar cancro colorectal de baixa mutação, mas não está restrito a este tipo de cancro. Contrariamente a outros pipelines, o nosso abrange todos os passos necessários para transformar amostras de tecido sequenciado em previsões de neoantígenos. Além disso, este usa a informação do ácido ribonucleico para confirmar que as mutações específicas do tumor são expressas. O pipeline é de código aberto, gratuito e foi desenhado com reprodutibilidade, modularidade e colaboração em mente. Adicionalmente, este trabalho inclui um módulo de visualização de dados através do qual é possível visualizar os resultados do pipeline. O pipeline apresentado é uma revisão integral do pipeline previamente utilizado pelo grupo de Imunogenómica do Centro Médico Universitário de Leiden, alcançando uma redução do tempo de execução de dias para horas e mantendo a identificação de neoantígenos em linha com os resultados anteriores.

# Palavras Chave

Identificação de neoantígenos; Pipeline bioinformático; Vacina personalizada contra o cancro; Sequenciação de nova geração; Computação de alto desempenho

# Contents

# List of Figures

# List of Tables

# Listings

# Acronyms

**A** Adenine

**ANN** Artificial Neural Network

**API** Application Programming Interface

**AUC** Area Under the Curve

**BAM** Binary Alignment Map

**BLAST** Basic Local Alignment Search Tool

**bp** base pair

**BQSR** Base Quality Score Recalibration

**C** Cytosine

**cDNA** complementary DNA

**CI** Continuous Integration

**CLI** Command-Line Interface

**CPU** Central Processing Unit

**CRC** Colorectal Cancer

**CTL** Cytotoxic T Cell

**CWL** Common Workflow Language

**DB** Database

**DNA** Deoxyribonucleic Acid

**EMBL-EBI** European Bioinformatics Institute

**FFPE** Formalin-Fixed Paraffin-Embedded

**FN** False Negative

**FP** False Positive

**G** Guanine

**GATK** Genome Analysis Toolkit

**GB** Gigabyte

**GRC** Genome Research Consortium

**GRCh37** Genome Research Consortium human build 37

**GRCh38** Genome Research Consortium human build 38

**GUI** Graphical User Interface

**hg19** Human genome build 19

**hg38** Human genome build 38

**HLA** Human Leukocyte Antigen

**HOCON** Human-Optimized Config Object Notation

**HPC** High-Performance Computing

**HTML** Hypertext Markup Language

**IGV** Integrative Genomics Viewer

**I/O** Input/Output

**JRE** Java Running Environment

**JSON** JavaScript Object Notation

**JVM** Java Virtual Machine

**KB** Kilobyte

**LUMC** Leiden University Medical Center

**MHC** Major Histocompatibility Complex

**MNV** Multi-Nucleotide Variant

**MUT** Mutant

**NFS** Network File System

**NGS** Next-Generation Sequencing

**OGS/GE** Open Grid Scheduler/Grid Engine

**OS** Operating System

**PGV-001** Personalized Genomic Vaccine 001

**PPV** Positive Predictive Value

**QC** Quality Control

**REST** Representational State Transfer

**RNA** Ribonucleic Acid

**RNA-Seq** RNA-Sequencing

**ROC** Receiver Operating Characteristic

**SASC** Sequencing Analysis Support Core

**SGE** Sun Grid Engine

**SNV** Single Nucleotide Variant

**SSH** Secure Shell

**SV** Structural Variant

**T** Thymine

**TCR** T Cell Receptor

**TN** True Negative

**TP** True Positive

**TPR** True Positive Rate

**TNR** True Negative Rate

**U** Uracil

**UI** User Interface

**UML** Unified Modeling Language

**URL** Uniform Resource Locator

**VAF** Variant Allele Frequency

**VEP** Variant Effect Predictor

**VCF** Variant Call Format

**WDL** Workflow Description Language

**WES** Whole-Exome Sequencing

**WFMS** Workflow Management System

**WGS** Whole-Genome Sequencing

**WT** Wild Type

**XML** Extensible Markup Language

**YAML** YAML Ain't Markup Language

# 1

# Introduction

## Contents

## 1.1 Context and Motivation

Computational methods aid researchers in finding new therapeutics for cancer treatment. Next-Generation Sequencing (NGS) allows the sequencing of Deoxyribonucleic Acid (DNA) and Ribonucleic Acid (RNA) much faster and cheaper than was previously possible [2]. *Sequencing* is the process of determining the nucleotide order of DNA or RNA extracted from cells, which in our context are human cells. *Nucleotides* are the building blocks of DNA and RNA and consist of a sugar molecule (either ribose in RNA or deoxyribose in DNA) attached to a phosphate group and a nitrogen-containing base. There are 5 nucleotide *bases* which give nucleotides their name: Guanine (G), Cytosine (C), Adenine (A), Thymine (T) and Uracil (U); T is specific to DNA and U to RNA. When these bases bind to each other they form *base pairs*. The entire set of an individual's DNA constitutes their *genome*. Figure 1.1 is a high-level representation of the structure of DNA molecules using the previous concepts.

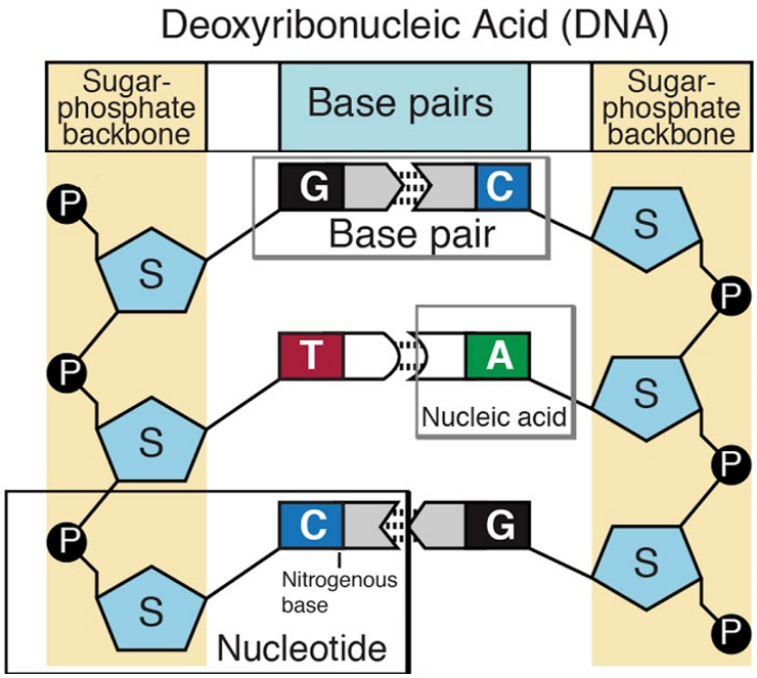


**Figure 1.1:** DNA molecule structure, where S and P represent sugar molecules and phosphate groups, respectively. Adapted from [3].

Inside our cells DNA is *transcribed* into RNA, which in turn is *translated* into a protein. During transcription RNA *splicing* occurs, in which *introns* are removed and *exons* remain. Because introns are excluded before the creation of proteins, they are *non-coding* parts of the genome as opposed to exons which are *coding*. Being coding means that that portion of nucleotides generates a protein. Figure 1.2 summarizes the concepts in this paragraph.

Depending on factors such as the sequencing system used and the percentage of the genome covered, sequencing a human tissue sample can result in billions of *reads* [4, 5] – sequences of nucleotides

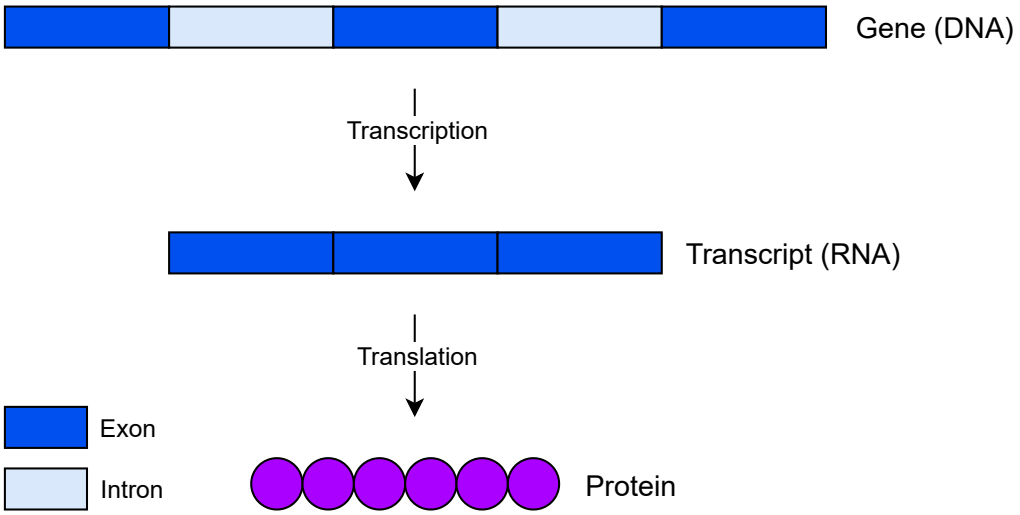


**Figure 1.2:** Transformation of DNA into protein through transcription and translation.

each corresponding to some portion of the genome. An important concept is *depth* or *coverage* which represent the number of unique reads that cover a given nucleotide at a certain position as shown in Figure 1.3. These concepts can be generalized to have as a reference point a *locus* (a site on a chromosome) or the whole genome instead of a single position. Having higher depth is desired as it helps distinguish between sequencing errors and real nucleotide variation.

```
Read 1 | ATCCTACTTATACTCCA
Read 2 |    CTACTTATACTCCACGG
Read 3 |        TTATACTCCACGGGCGT
-------+-------------------------
Depth  | 1112222333333333222111
```

**Figure 1.3:** Example of an overlap of sequencing reads, with the depth indicated at each position.

Illumina's NGS technologies [6] are currently the most widely used in sequencing due to having a good balance between quality and cost. Illumina commercializes various models of *sequencing platforms*, i.e., machines that run the sequencing analysis on the input samples, such as tumor tissue.

Based on the sequencing data of several individuals, a human *reference genome* was created by the Genome Research Consortium (GRC) [7]. This reference genome is used by the scientific community as a point of comparison in NGS analyses. The way nucleotides are arranged in each individual's genome determines their characteristics and a portion of the genome that codes for a protein is known as a *gene*. *Alleles* are variations of a given gene that can range from single nucleotide differences to more

complex recombination events. Any change in the DNA sequence relative to a reference genome is a *variant*. Variants in the coding part of a genome, i.e. genes, are known as coding variants. *Mutations* are variants that have deleterious or advantageous effects in cells and are rare events. In cancers, mutations on genes that control cellular growth often accumulate, leading to an uncontrolled proliferation of cancer cells. Variants in tumor cells that change protein-coding sequences may generate neoantigens. *Antigens* are substances that induce an immune response on our body, and *neoantigens* are antigens encoded by *tumor-specific variants*, i.e, variants not present in normal (healthy) tissue. Neoantigens have been shown to play a significant role in mediating the destruction of tumor cells by the immune system [8–10]. Tumors with a higher number of tumor specific-mutations, and consequently more neoantigens have a better prognostic [11]. *T cells* are a type of white blood cells called *lymphocytes* and are fundamental to the body's immune response. Tumor cells are killed through the action of Cytotoxic T Cells (CTLs) that recognize neoantigens that bind to the T Cell Receptor (TCR) and are presented by cell surface Major Histocompatibility Complex (MHC) molecules in tumor cells (see Figure 1.4). As such, numerous studies have targeted patient-specific neoantigens to clear tumors with promising results [12, 13]; this treatment can be done through the administration of neoantigen vaccines – see Figure 1.5 for a visualization of the results in a melanoma patient. This type of customized treatment to patients fits into the category of *personalized medicine*, to which the ever-decreasing time and sequencing costs have contributed to making it more viable in medical practice. Furthermore, the specific region on an antigen that the CTLs recognize is called an *epitope*, and a *neoepitope* is an epitope encoded by a tumor-specific variant.

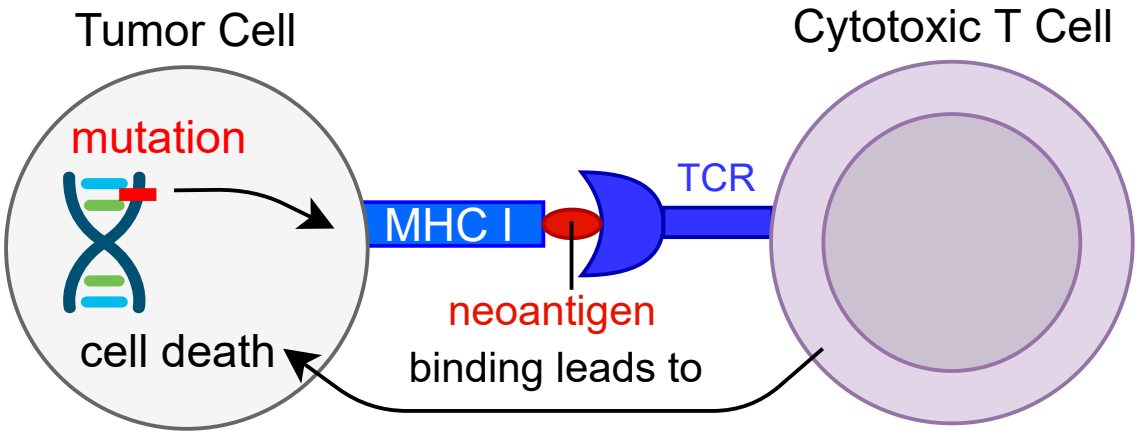


**Figure 1.4:** Elimination of a tumor cell with a non-synonymous coding mutation by a CTL. This mutation originated a neoantigen that is then presented by a cell surface MHC molecule. The CTL then specifically recognizes the presented neoantigen and causes the tumor's cell death.

In practice, to identify neoantigens NGS data must go through several steps of analysis and transformation which can be accomplished using a bioinformatics *pipeline*: a series of software steps, usually chained together using a framework that controls the process flow. The framework can be as basic as

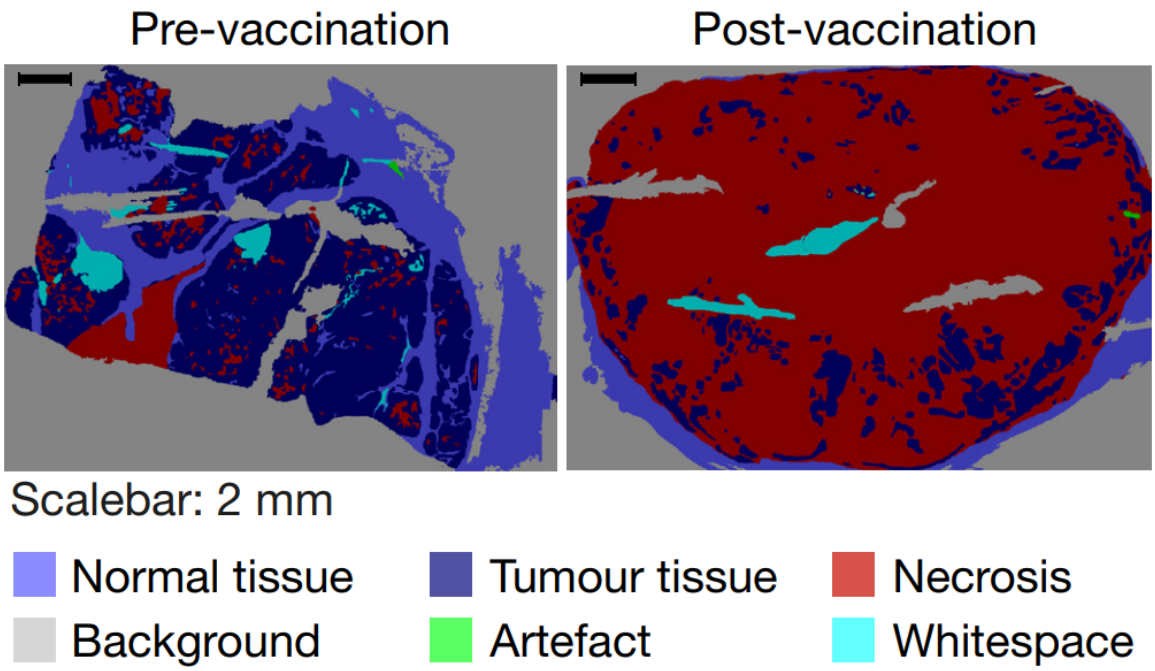


**Figure 1.5:** Computer visualization of melanoma cells, before and after the patient was treated with a neoantigen vaccine. We can observe that after vaccination the majority of the cells were seen to be dying (necrosis). Reproduced from [14].

a Unix shell script but preferably a Workflow Management System (WFMS) is used. A proper WFMS enables the pipeline to continue where it left off if interrupted, has the ability to make use of multiple computing nodes, and handles job scheduling without requiring modifications to the existing code. Another good practice is to use containers to house the software being run at each step of the pipeline workflow, which allows for reproducibility and portability of the pipeline, important aspects especially as its complexity increases. Figure 1.6 represents an example flow of a pipeline executing in a High-Performance Computing (HPC) cluster environment using a WFMS framework and software containerization.

As a final consideration, due to the complex nature of these pipelines in terms of the numerous inputs and transformations that the data goes through, visualizing the outputs is valuable because it helps detecting errors such as malformed or mismatching inputs, or incorrect bioinformatics software configuration, thus avoiding drawing wrong conclusions about the processed data.

## 1.2 Problem Formulation

This dissertation presents an example of how the branch of bioinformatics can positively impact cancer research with the development of a software pipeline that is capable of processing a large amount of sequencing data from patients and with it identify neoantigens that can then be used by researchers to develop targeted cancer vaccines. In particular, we will be analyzing sequencing data from Colorectal

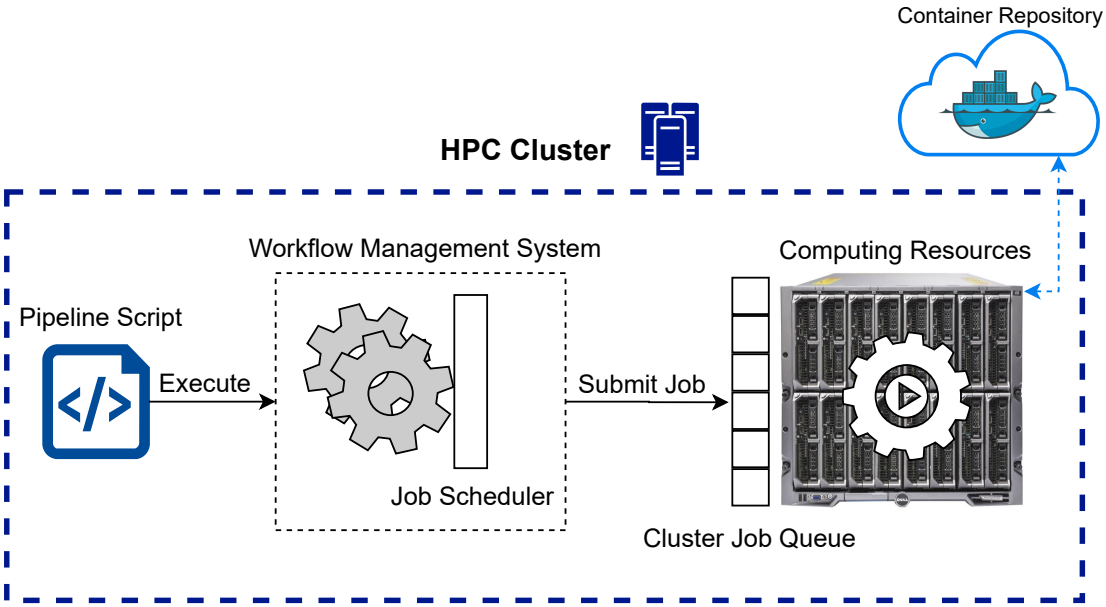


**Figure 1.6:** Example process flow of a pipeline executing in an HPC cluster following good practices. A WFMS is used to create and submit the jobs to the cluster job queue without having to explicitly write code in the pipeline script to do so. Also, containerization of the software, which provides reproducibility and portability to the pipeline, is depicted in the upper right corner of the figure by a cloud with the Docker logo [15]. It is depicted in this way to show that the containerized software executed by the pipeline is fetched from a repository hosted in the cloud.

Cancer (CRC) with low mutation numbers, with the key aspect being the identification of all tumor-specific variants that are coding non-synonymous, i.e., variants that cause changes to the proteins' sequence. This analysis was already being done by the Immunogenomics group of the Department of Pathology of Leiden University Medical Center (LUMC) [16], environment where and for which the pipeline was developed. The main issue with the pipeline that was in place prior to the development of the one presented in this dissertation was that its execution time was in the order of days, something that could be vastly improved by leveraging the available computational power of the HPC cluster at LUMC. However, the previous implementation of the pipeline was not deployable in the cluster, hence the need for a complete overhaul.

## 1.3 Contributions

The main contributions of this work, put into context with the solution that was previously in place, are:

- An integrated pipeline [17] (currently in release 4.0.0 [18]) that adheres to recent standards in the bioinformatics field, implemented to run using the Cromwell WFMS [19] as a way to utilize HPC clusters.

- A reduction in the execution time from days to hours when compared to the previously used pipeline.
- A readable, modular and reusable implementation of the integrated pipeline achieved with the Workflow Description Language (WDL) [19] (see Sections 3.3 and 3.6).
- Reproducibility as a consequence of software being containerized in the integrated pipeline (see Section 3.5).
- Improved filtering of the predicted variants using data from other patients by implementing more recent variant calling software functionalities into the pipeline (see Section 3.8.3.A).
- An overhauled Human Leukocyte Antigen (HLA) binding prediction module with a novel method for filtering predicted neoantigens by using data available in public and continuously updated databases (see Section 3.9.3).
- A separate R Shiny [20] user interface for visualizing the results of a crucial step of the pipeline, allowing to spot errors (see Section 3.10).
- Software that is open source, publicly available in GitHub and designed to be used by others.

## 1.4 Thesis Outline

Following this introductory first chapter, this dissertation begins by giving an overview of the typical workflow for identifying neoantigens in Chapter 2. It comprises the pieces of available software and their evaluation, alongside the necessary biological context to motivate each step of the workflow. In the same chapter we also address data visualization and related pipelines, including the one that was previously in use at LUMC.

Chapter 3 is dedicated to the developed pipeline, covering the technologies involved, such as the containerization software used, the Cromwell WFMS, each pipeline module, visualization of the results with R Shiny, and the Continuous Integration (CI) and testing methodologies.

In Chapter 4, we address some of the problems found and the validation of the results of the new pipeline to guarantee at least the same predictive performance as the previous pipeline.

Finally, Chapter 5 is reserved to conclusions drawn from this work and possible improvements.

# 2

# Background: Typical Neoantigen Identification Workflow

## Contents

## 2.1 Overview

In this chapter, we start by providing an overview of the computational methods and the typical workflow [21] for identifying neoantigens and then discuss it in more detail. Each step of the workflow is accompanied by a basic biological context, since it helps to reason about the task at hand. Figure 2.1 summarizes the workflow: tumor and normal tissue have their DNA extracted (also RNA in the case of the tumor) and sequenced. Then comes identification of the patient's HLA type and, independently, Quality Control (QC), adapter trimming and read mapping are performed. In the latter, we take millions of independent nucleotide sequences (reads) and have them mapped to their corresponding position in the reference genome. After the reads are aligned, *variant calling* is performed, i.e., nucleotide differences between the aligned tumor and normal reads are identified. Using the RNA mapping information it is possible to confirm that a variant is expressed. Finally, for all tumor-specific variants present in the RNA sequences, binding prediction to the patient's HLA type is done.

## 2.2 Turning Tissue into Strings

Numerous steps are necessary to transform raw sequencing data into complete genes sequences or genomes. While many tools are available, which ones to use depend on several factors, one being the sequencing platform used.

On this project, we will focus only on Illumina's platforms. Each platform has different technical specifications such as accuracy and error rate, as well as *read length* which influences the amount of nucleotides from a given DNA or RNA fragment that are sequentially read, i.e., that end up in the same read. The read length has implications downstream in the analysis, particularly in the alignment step as the precision of an alignment is generally better with longer reads because there is more contextual information (more contiguous nucleotides) to help matching a certain read to a genome region. The read length of a sequence platform has to be taken into account when creating the DNA or RNA fragments to be sequenced. Otherwise, an inner portion of a given fragment will not be sequenced when *paired-end sequencing* is done (see Figure 2.2). In paired-end sequencing, both ends of the fragment are read: read 1 (R1) and read 2 (R2). This contrasts with *single-end sequencing* where only one end of the fragments is sequenced.

Another factor that influences tool choice is the quality of the input material. There are different ways to prepare and preserve samples, e.g., Formalin-Fixed Paraffin-Embedded (FFPE) and frozen fresh tissue. In general, FFPE material has low quality which can result in a higher percentage of sequence artifacts, i.e., DNA sequence changes that in reality are not present in the original material. Low quality material is often more fragmented (smaller insert sizes) resulting in an inferior genome coverage. The choice of tools to use depends as well on the portion of the genome targeted. This can be the entire

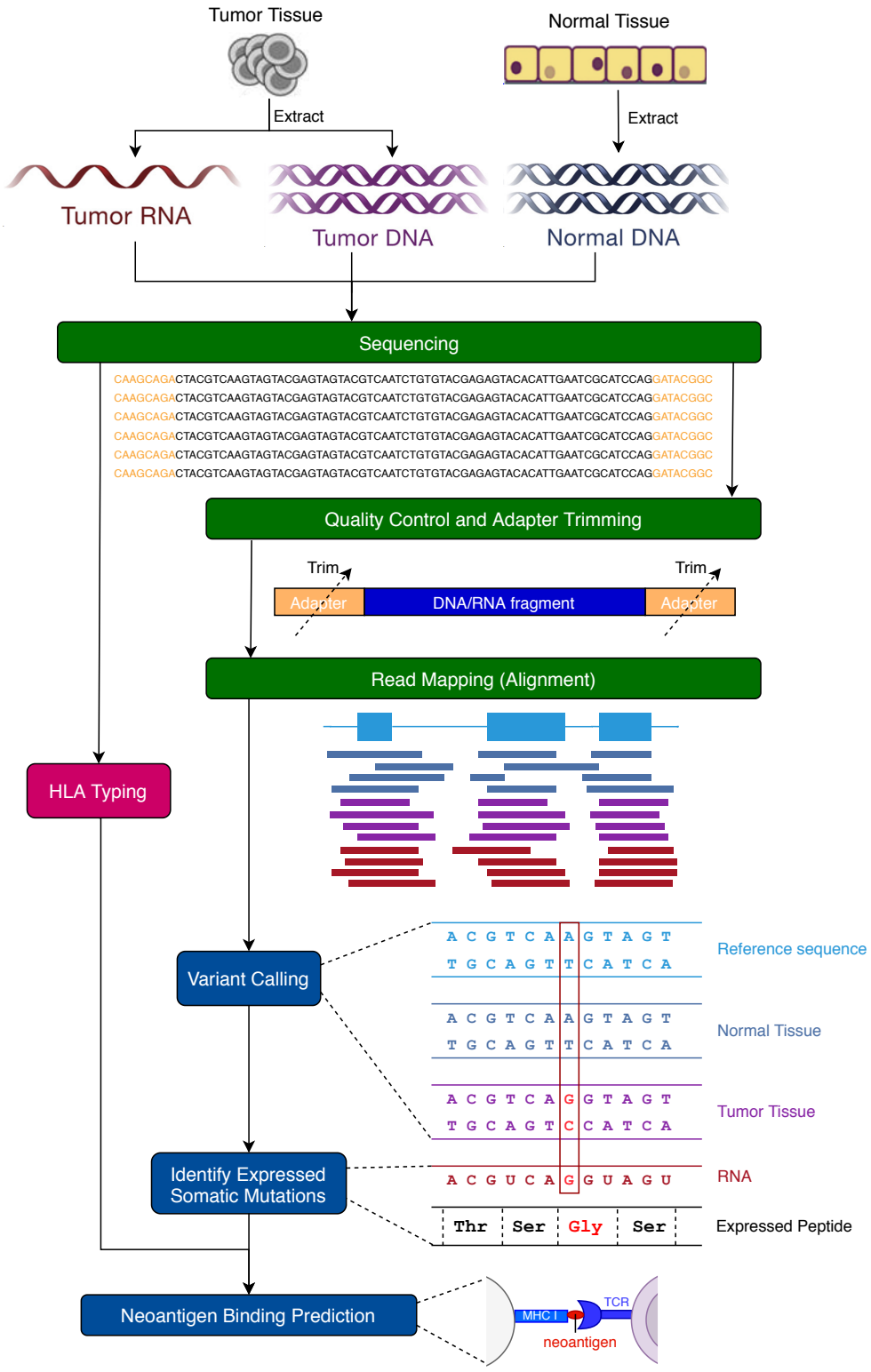


**Figure 2.1:** Typical neoantigen identification workflow. The steps highlighted in green transform the tissue samples' sequencing data into a format that is compatible with the tools capable of performing the analyses required for neoantigen binding prediction, highlighted in blue.

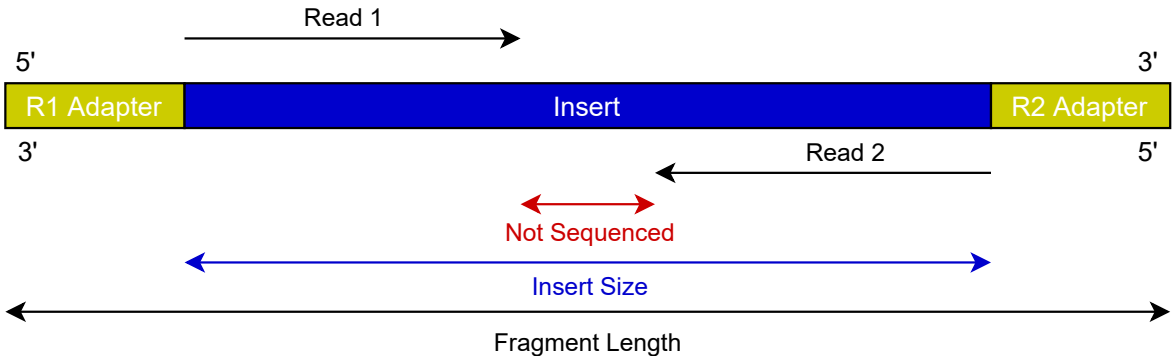


**Figure 2.2:** DNA fragment with an insert size longer than the length of both reads.

genome, only the exome, only the transcriptome, or only a small portion of the genome such as a single gene, which are called as Whole-Genome Sequencing (WGS), Whole-Exome Sequencing (WES), RNA-Sequencing (RNA-Seq), and targeted-sequencing, respectively. Lastly, in our context of building pipelines to run in HPC clusters, there are usually software or hardware resources restrictions in place. Nowadays, the former can be overcome by using containerization tools and the latter usually tackled by tuning the software parameters.

A useful and regularly updated resource to help deciding which tools to use is the Broad Institute's Genome Analysis Toolkit (GATK) Best Practices [22]. A complete resource not only for the steps in this section, but also for variant discovery and filtering, which are covered later in this report. GATK Best Practices focus largely on human (WGS and WES) samples sequenced with Illumina technology.

### 2.2.1 Sequencing

In the context of this work, sequencing is the process of determining the nucleotide order of a given DNA or RNA fragment. The necessary steps to transform the biological material – in our case, normal and tumor tissue – into DNA or RNA libraries are not the main focus of this report. Nevertheless, it is important to briefly explain sequencing because the pipeline is dependent on its correctness. Generally, the laboratorial steps (summarized in Figure 2.3 for DNA) are:

1. Extract genomic DNA or RNA from the source material and fragment it. Specifically for RNA, generate complementary DNA (cDNA) through reverse transcription (refer back to Figure 1.2). In sequencing, cDNA is then handled identically to DNA. Select fragments of the desired size and add adapters. *Adapters*, shown at both ends of the DNA fragment in Figure 2.2, are predetermined short synthesized nucleotide sequences that serve as markers to identify the DNA sequences of interest, called the inserts, shown in Figure 2.2.

2. *Amplify* (create thousands of copies of) the fragments.

3. Finally, sequencing is done using an Illumina sequencing platform – in our case the HiSeq platform was used.

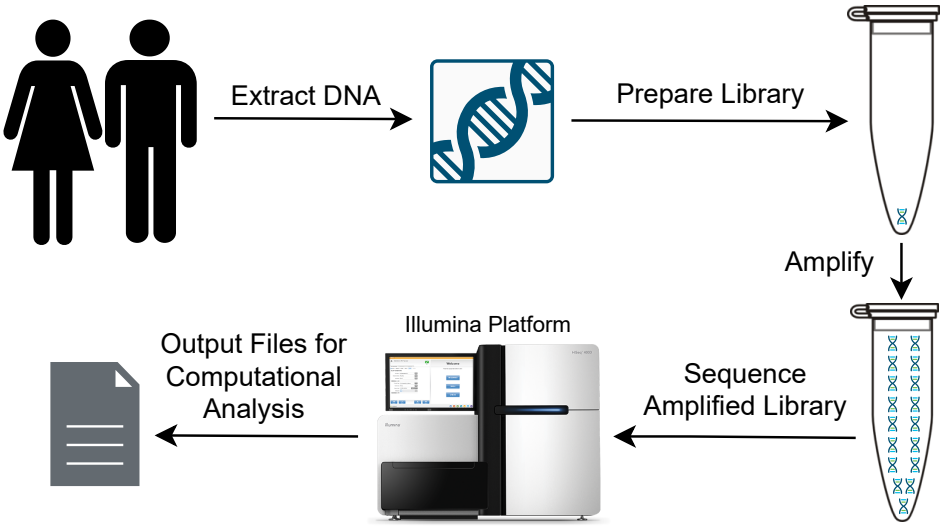


**Figure 2.3:** DNA sequencing steps

The resulting data from sequencing are FASTQ files, the standard format for storing biological sequences. These files (exemplified in Listing 2.1) consist of nucleotide sequences with their corresponding quality scores, each represented by an ASCII character. They can be multiple GB in size, containing millions of lines.

**Listing 2.1:** FASTQ file excerpt representing a read sequence. Description by line number: 1) sequence identifier; 2) nucleotide sequence; 3) optional quality score identifier (not present) preceded by a "+" sign; 4) quality score of each nucleotide.

```
1 @K00296:19:HC2YTBBXX:2:1101:1600:1103 1:N:0:ATCACG
2 NCTGCAGTGCAGTGACTATATTCTTCACAATAATCAAACNGGTCANTTATCGCCTACACCTCNTGTTTGACAAAG
3 +
4 #AAFFAAJJJJJFAJJFJFJJJJJFJJJJFJFFJJFJJJ#JFFJJ#7<JJJJJJJFJJF-A#A7<<<JJJJJA
```

### 2.2.2 Quality Control and Adapter Trimming

After sequencing, it is necessary to remove potential biases and technical errors. FastQC [23] can be used to assess the quality of the sequencing data, by providing an in-depth report on data quality, such as base quality, GC content (i.e., percentage of bases that are G or C), and overrepresented sequences such as adapters and *primers* used during preparation of DNA or RNA libraries. Figure 2.4 shows the

"Per base sequence quality" plot generated by FastQC. This plot allows the identification of poor libraries as these will have reads with low quality.

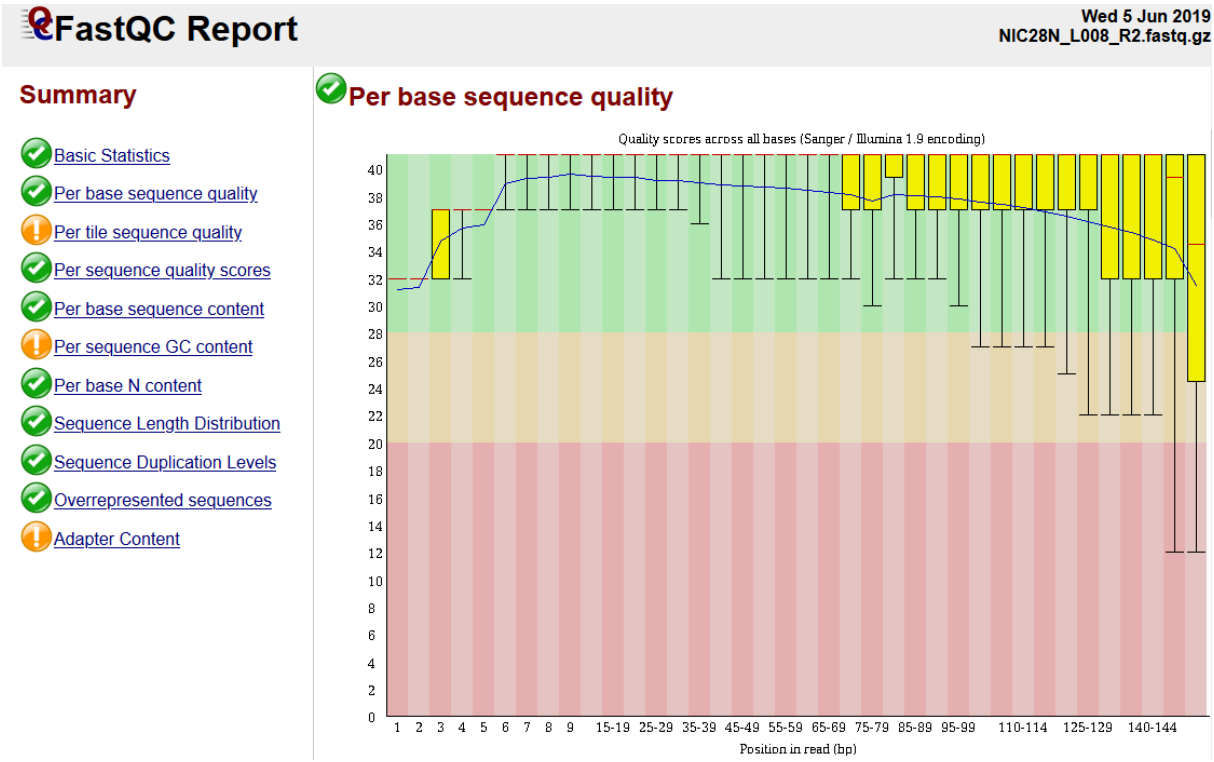


**Figure 2.4:** Example FastQC report for Illumina sequence data with the "Per base sequence quality" plot in focus. The vertical axis represents base quality measured in Phred quality scores and the horizontal axis represents the nucleotide position in read. Green is considered good quality, yellow average and red low. On the left, links to the other automatically generated plots are available.

Base quality is measured using Phred's quality scores. These are calculated with:

$$q = -10 \times \log_{10}(p)$$

where $q$ is the quality score and $p$ the estimated error probability for that base (calculated by the program Phred) [24]. Table 2.1 shows quality score values that translate to intuitive probabilities of incorrect base identification. Although a Phred score of 30 might seem reasonable, taking into account that DNA comprises around 3 billion bases, 99,9% base accuracy means we would be left with potentially 3 million incorrect base calls in a WGS analysis.

Low-quality portions of reads, for instance, end-specific poor quality bases, can be trimmed. However, short reads result in lower mapping (covered in section 2.2.3) accuracy and sequencing depth. As was shown in an extensive evaluation of read trimming effects [25], there is no definitive answer for what the best trimming algorithm is since it is highly dependent on the dataset, downstream analysis, and trade-offs resulting from the parameters chosen by the user. However, if the main purpose is adapter

**Table 2.1:** Phred's base-specific quality scores

| Phred Quality Score | Probability of Incorrect Base | Base Accuracy |
|---|---|---|
| 10 | 1 in 10 | 90% |
| 20 | 1 in 100 | 99% |
| 30 | 1 in 1000 | 99.9% |
| 40 | 1 in 10,000 | 99.99% |
| 50 | 1 in 100,000 | 99.999% |
| 60 | 1 in 1,000,000 | 99.9999% |

contamination removal, the Cutadapt [26] tool provides a relatively conservative trimming feature. After trimming, quality control should be run a second time to verify that sequence quality has improved.

### 2.2.3 Read Mapping (Alignment)

Upon removing low quality bases, reads are mapped against a reference genome. The commonly used versions of the human genome reference are the Genome Research Consortium human build 37 (GRCh37) and Genome Research Consortium human build 38 (GRCh38), also referred to as Human genome build 19 (hg19) and Human genome build 38 (hg38), respectively. Besides the latter being more recent, a notable difference between these two versions is that the hg38 reference is comprised of considerably more alternate *contigs* (contiguous nucleotide sequences, such as an entire chromosome). Read mapping consists of aligning the reads to the reference genome using a *read aligner*, i.e., a software that aligns nucleotide sequences as illustrated in Figure 2.5. Read mapping can be thought of as assembling a puzzle of the reference genome, where the puzzle pieces are the sequenced reads. The caveat is that, due to normal variation, the reads often do not exactly match the reference genome, thus constituting a challenge for the mapping algorithms. Essentially, read alignment is an approximate string matching problem.

There are two major algorithmic ideas to read alignment: filtering and indexing [27]. Filtering excludes large regions of the reference genome that cannot contain an approximate match, usually resorting to the pigeonhole or the $q$-gram lemma. Indexing comprises preprocessing the reference, the set of reads, or both in an elaborate way that avoids scanning the entire reference, thus allowing for faster querying at the expense of larger memory consumption. The string indices used are suffix arrays [28], enhanced suffix arrays [29], and the FM-index [30] (based on the Burrows-Wheeler transform [31] and auxiliary tables). The enhanced suffix array and FM-index can determine in linear time with respect to the length of the query if a query sequence occurs in the reference.

For DNA sequence data, the most commonly used programs for read mapping are BWA [32] and Bowtie 2 [33], whereas for RNA sequence data STAR [34], TopHat2 [35] and Kallisto [36] are common choices. The aforementioned aligners use the FM-index, except Kallisto which uses a colored de Bruijn graph [37] as an index. Note that for RNA data, an aligner that is capable of handling splice

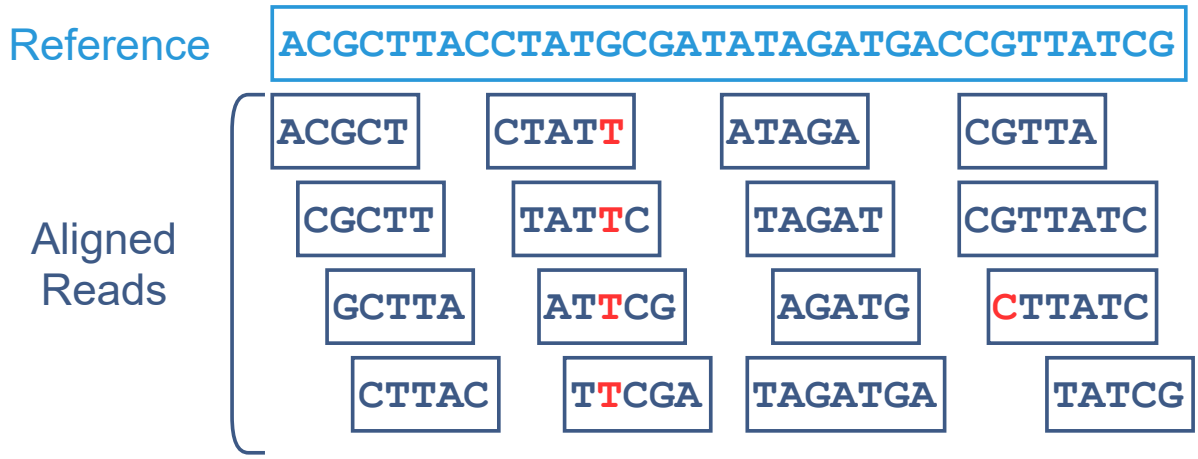


**Figure 2.5:** Short reads aligned to a reference genome. Variations are shown in red, exemplifying what could be a SNV (from G to T) and a read error (C instead of G). The size of the reads is merely illustrative as their sizes can vary greatly depending on the sequencing step of the pipeline.

junctions (exon-intron boundaries) is needed. A comprehensive list of read mappers from the European Bioinformatics Institute (EMBL-EBI) [38] is available online [39]. Although it was last updated on the 27th December 2017, it is still relevant today. Lastly, read mapping software outputs files in the Binary Alignment Map (BAM) format which can be inspected and post-processed using Command-Line Interface (CLI) tools such as SAMtools [40], BAMtools [41] and Picard [42]. BAM files can be multiple GB in size.

## 2.3 Somatic Variant Calling: Finding Nucleotide Differences in Tumors

Variant calling is the process of identifying alterations in the genetic sequences, i.e., locating different nucleotides other than the expected at a certain position relative to the reference. In *somatic variant calling*, cancer and matched normal (derived from normal tissue of the same patient) sequences are compared in order to detect *somatic variants*, i.e. variants present in the tumor but absent from normal cells. This step is key to the success of the pipeline: each variant that goes undetected is a lost chance to identify an altered peptide. These novel peptides, exclusive to cancer cells, may serve as neoantigens mediating the destruction of tumor cells by the immune system.

Most current somatic variant callers are designed to analyze matched tumor and normal sequences from the same patient simultaneously. The fundamental idea is to pinpoint potential variants in the tumor sequence (by comparing it to the reference), while at the same time identifying whether they are somatic or *germline variants* (present both in tumor and normal cells) by comparing them with the matched normal sequence [43]. Figure 2.6 presents the latter idea by highlighting in red the variant nucleotides

in the tumor tissue sequence relative to the reference and that are not present in the normal tissue, i.e., that are not germline.

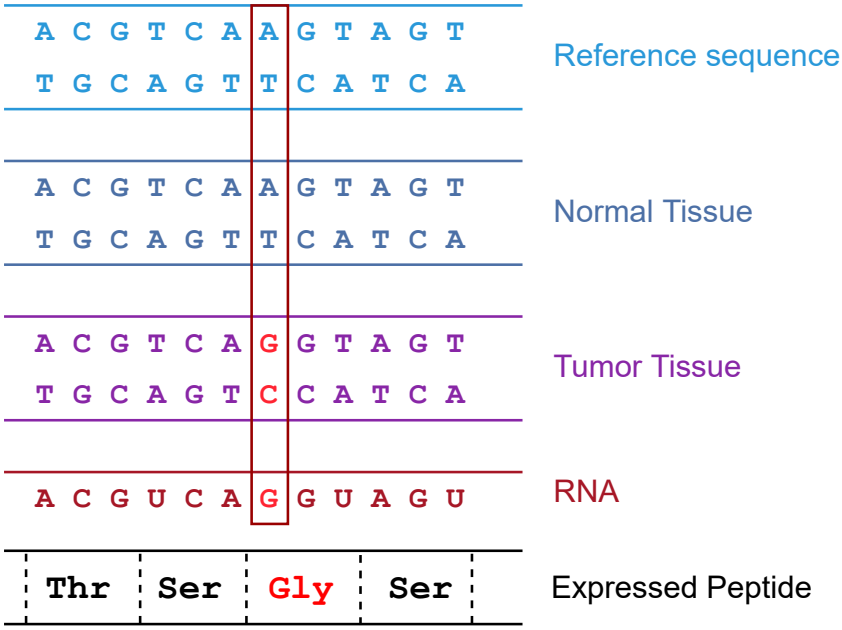


**Figure 2.6:** The idea behind somatic variant calling is to compare the nucleotide sequence of the tumor with that of the reference in order to find variations. Simultaneously, it is necessary to confirm that a given variant is not germline by also taking into account the matched normal sequence.

Germline variants are inherited from parents, whereas somatic variants consist of acquired differences during cell division and can be caused due to environmental insults and lifestyle risk factors, such as tobacco use [44]. There are a number of differences between somatic and germline variant calling. Whereas germline variants are expected to have a frequency of around 50% or 100%, depending on 1 or 2 alleles being different relative to the reference sequence, respectively, in tumor tissue variant frequency distribution falls in a continuous range. This is due to normal and tumor tissue admixture; *subclonal variations* (as tumor cells replicate, they can produce clones that are further mutated); *copy number variations*, the phenomenon in which sections of the genome are repeated resulting in variable number of copies of that genome segment; or loss of *heterozygosity*, a common genetic event in cancer by which at least one copy of the allele is lost, leading to part of the genome appearing homozygous in the tumour where heterozygous in matching normal DNA [45]. To understand this concept we have to take into account that humans are *diploid* organisms, which means that each cell contains two copies of each chromosome. Consequently, humans also have two copies of each gene, one in each chromosome. If the alleles corresponding to the same gene match in both chromosomes, the person is *homozygous*, otherwise, *heterozygous* for that allele.

There are three categories of variants: Single Nucleotide Variant (SNV), *insertion* or *deletion* (*indel*), and Structural Variant (SV). Few variant callers are versatile enough to call all three types because

fundamentally different algorithms are required for each [43].

There are three major strategies adopted by variant callers: 1) position-based: comprises heuristic, joint genotype, and joint allele frequency analysis. 2) Haplotype-based: assembles reads locally in a region and generates candidate *haplotypes* (linked SNVs that tend to always occur together). 3) Machine learning based. Example tools that fit these three categories are Strelka [46], MuTect2 [47], and SomaticSeq [48], respectively. To choose a variant caller (or a combination of them), there are a number of factors to take into account [43].

The most important factor is the type of variants to be called: SNVs, indels or SVs. The majority of the available variant callers report SNVs, yet only some report indels, SVs, or both. Hence, if besides SNVs indels and SVs are of interest, for convenience, one can use all-around tools like Platypus [49], Seurat [50], and VarDict [51]. However, the rationale for the choice of a variant caller should go beyond convenience, as it is a crucial step for the success of neoantigen detection.

Another aspect in choosing a variant caller is the Variant Allele Frequency (VAF) threshold, i.e., the minimum frequency to consider a variant true and not a sequencing artifact. Variant callers based on joint genotype analysis, such as SomaticSniper [52], FaSD-somatic [53], Virmid [54], JointSNVMix2 [55], SNVSniffer [56], and Seurat are not sensitive enough to detect low-frequency variants.

In case low-frequency variant calling is desired, particularly with high-coverage sequencing data, a caller that directly models allele frequencies should be selected. Examples of such callers are: Strelka, MuTect [57], LoFreq [10], EBCall [58], deepSNV [59], LoLoPicker [60], and MuSE [61].

Low-frequency variant calling accuracy has been analysed in Strelka and MuTect's articles [46, 57], and independently benchmarked [62, 63]. Xu et al. [62] compared sensitivity (true positive rate), and specificity (true negative rate) between well-established somatic variant callers such as Strelka, MuTect, and VarScan2 [64]. Strelka and MuTect were found to have achieved significantly higher sensitivity at the lowest VAF with similar or lower false positive rates than the other methods. Moreover, Strelka achieved considerably better sensitivity than MuTect under the recommended settings, but at the expense of a much higher false positive rate [62]. It was not clear whether further tuning of these two algorithms would change the latter observation's validity. A relevant note is that a combination of different variant callers resulted in more SNVs being detected. In Rubinsteyn et al. [65], an example of a computational vaccine pipeline, MuTect and Strelka were used in conjunction. This pipeline is called the Personalized Genomic Vaccine 001 (PGV-001) pipeline, and it shares many similarities with the steps and end goal of the work here presented. Another point concerning low-frequency variant calling is that heuristic-based analysis callers are also capable of achieving good accuracy if the VAF thresholds are carefully chosen, e.g., 1% for VarDict [63] and less than 5% for VarScan2 [66].

Input data may also limit the choice of variant caller. Most callers take as input aligned reads of matched tumor-normal samples in the BAM file format but some require additional data such as a cohort

of control samples to obtain site-specific error rates, e.g., LoLoPicker, a list of SNVs called by other algorithms to perform haplotype analysis, e.g., LocHap [67], or variant calls from other somatic variant callers, e.g., SomaticSeq [48].

Variants called are written into Variant Call Format (VCF) files. Being a verbose file format and typically having thousands of lines, it is better handled with either a CLI tool such as VCFtools [68] or BCFtools [69], or with a genome visualization tool such as IGV (covered in Section 2.7). Listing 2.2 shows a simplified excerpt taken from a VCF file generated by Mutect2. Essentially, VCF files are composed by a header section with metadata, followed by a body – divided into (tab separated) columns – that contains the information regarding the variants. The metadata in the header comprises, for example, the description of the fields found in the body of the file, the variant calling software command that was run to generate the file, and the regions of the genome where the variants were called.

**Listing 2.2:** Adapted excerpt of a VCF file output by Mutect2. The metadata header lines are preceded by "#". In the body, the variant call with a "PASS" filter is predicted by Mutect2 to be a true variant, whereas the others are considered filtered. The "FORMAT" column – AD:AF:DP – indicates the format of the columns that follow it.

```
1  ##fileformat=VCFv4.2
2  ##FILTER=<ID=germline,Description="Evidence indicates this site is germline,
       not somatic">
3  ##FILTER=<ID=panel_of_normals,Description="Blacklisted site in panel of
       normals">
4  ##INFO=<ID=GERMQ,Number=1,Type=Integer,Description="Phred-scaled quality
       that alt alleles are not germline variants">
5  ##INFO=<ID=MBQ,Number=1,Type=Integer,Description="median base quality">
6  ##INFO=<ID=PON,Number=0,Type=Flag,Description="site found in panel of
       normals">
7  ##INFO=<ID=SEQQ,Number=1,Type=Integer,Description="Phred-scaled quality that
       alt alleles are not sequencing errors">
8  ##FORMAT=<ID=AD,Number=R,Type=Integer,Description="Allelic depths for the
       ref and alt alleles in the order listed">
9  ##FORMAT=<ID=AF,Number=A,Type=Float,Description="Allele fractions of
       alternate alleles in the tumor">
10 ##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Approximate read depth
       (reads with MQ=255 or with bad mates are filtered)">
11 ##contig=<ID=chr1,length=248956422>
12 ##contig=<ID=chr2,length=242193529>
13 #CHROM  POS REF ALT FILTER  INFO    FORMAT  NORMAL  TUMOR
```

```
14 chr1    1312235 T   G   panel_of_normals    GERMQ=93;MBQ=30;PON;SEQQ=64
       AD:AF:DP    138,9:0.027:147 98,13:0.095:111
15 chr2    50531288    G   T   PASS    GERMQ=93;MBQ=30;SEQQ=93 AD:AF:DP
       161,0:6.142e-03:161 95,19:0.172:114
16 chr2    229271314   CGCG    C   germline    GERMQ=1;MBQ=30;SEQQ=1
       AD:AF:DP    11,0:0.071:11   3,1:0.323:4
```

Variant callers will unlikely achieve 100% accuracy because of the imperfect nature of sequencing data: biases in library preparation, sequencing errors, and mismapped reads cause false-positive variants to be called as well as variants to go undetected. In this work the aim is to maximize the number of variants detected even if that costs us a higher rate of false-positives. More variants detected will increase the chance of finding a mutant peptide that triggers an immune response. False-positive calls can later be filtered through downstream validation, both computational and laboratorial.

## 2.4 Predicting Mutant Proteins from Somatic Variants: Finding Tumor-specific Proteins

The main aim of this work is detecting somatic variants restricted to the tumor that result in different proteins than the ones produced by normal cells. For the purpose of identifying these proteins or their short peptides, it is not enough to predict a DNA variant without considering the *transcripts* (coding sequences) in which it occurs [65].

As shown in Figure 1.2, after DNA transcription to RNA, RNA splicing occurs leaving a sequence of exons, which represents a coding sequence. Many genes have multiple *isoforms*, i.e., alternative transcripts with potentially different functions. Furthermore, various somatic variants can co-occur in the same sequence. As such, RNA sequencing data is valuable and can be analyzed using a program such as Isovar [70], to determine the variant coding sequence, i.e., the novel coding sequence created by tumor cells. This has two implications: first, the transcript in which the variant occurs is predicted from the RNA-Seq data; and second, the adjacent variants are phased, i.e., variants that co-occur are identified, information without which we could potentially select two different vaccine peptides, instead of just the phased one.

In cases where RNA-Seq data is not available, the standard way to predict the altered protein sequence is to extract the sequence of the reference transcripts from a public database such as Ensembl [71] or GENCODE [72], which can be accessed programmatically in various ways. For example, using the Variant Effect Predictor (VEP) toolset [73] for Ensembl or the ANNOVAR [74] tool for both databases. Such tools not only retrieve the reference transcripts, but also identify whether variants

cause protein coding changes and the amino acids that are affected.

## 2.5 HLA Typing: Different Patients React Differently

The Human Leukocyte Antigen (HLA) complex, part of chromosome 6, codes for several proteins involved in immune system functions. The genes coding these proteins are categorized into HLA classes I, II and III. The most studied belong to classes I and II. The proteins encoded by these genes interact with immune cells to induce an immune response. Note that the HLA is equivalent to the MHC but is specific to humans. This gene complex is the most polymorphic region in the human genome, which means that it is rare to find the exact same alleles between two individuals. Approximately 40% of all the registered HLA alleles have only been observed once (in a single individual), being considered "very rare" [75]. Moreover, "common" alleles are those observed at frequencies greater than just 0.001 [76]. The homology and diversity observed in the HLA alleles makes HLA typing from sequencing data difficult. Nevertheless, typing is crucial in neoantigen binding prediction to enable assessing how strongly variant peptides bind to the patient's HLA molecule (see Section 2.6.1). Strong binders are more likely to be recognized by the immune system (illustrated in Figure 1.4).

Various bioinformatics tools have been developed to type HLA alleles of which the most accurate in typing HLA class I (and limited to it) is OptiType [77] [78]. However, when considering simultaneous HLA class I and class II typing, the most accurate software depends on the type of sequencing data [78]. Furthermore, to analyse the accuracy of different tools we have to take into account that some HLA typing software outputs more than one HLA allele prediction when the typing confidence is low. Considering only the top one prediction (the allele each software chose as most likely), for WGS HLA-VBSeq [79] is the most accurate. For WES and RNA-Seq it is PHLAT [80]. When considering the top three predictions for RNA-Seq data, Seq2hla [81] is as accurate as PHLAT. Moreover, Seq2hla was used in the aforementioned PGV-001 pipeline where its HLA typing output for tumor RNA across 10 samples only disagreed on a single HLA allele when compared to validation data obtained from sequencing of normal sample DNA. Although it can be noted that OptiType runs considerably slower than the other three tools, we favor accuracy. Additionally, only HLA class I was typed.

## 2.6 Predicting Peptide Immunogenicity: Will the Peptides Bind?

Immunogenicity is the ability of a substance, such as a neoantigen, to provoke an immune response. The recognition of a neoantigen by the immune system depends on several biological factors: peptide preprocessing, transportation to cell surface, HLA binding, followed by recognition and binding by the TCR on T cells (effector cells of the immune system). In this work, we do not need to go into the

biological details of these processes, but rather recognize their existence, so we can break down the analysis in several steps.

We will now be taking a look at some of the best performing tools related to predicting peptide immunogenicity. In the reviewed articles, the performance of prediction methods is usually assessed with the value of the Area Under the Curve (AUC) of a Receiver Operating Characteristic (ROC) curve. As such. to give a sense of prediction performance, the tool's analysis that follows is accompanied by AUC values – a value of 1.0 for the AUC represents a perfect prediction and 0.5 a performance equivalent to random assignments.

### 2.6.1 Predicting HLA Binding

The immune system is more likely to recognize a variant peptide if it is strongly bound to the patient's HLA molecule. Figure 1.4 shows this binding occurring in a tumor cell: a neoantigen (variant peptide), binds to the MHC molecule – the HLA in humans – and the immune system, of which the Cytotoxic T Cell is part of, recognizes the neoantigen and causes the tumor's cell death.

Binding affinity prediction to HLA class I is more accurate than for class II. The best performing algorithms for the former, NetMHC [82] and NetMHCpan [83], are based on Artificial Neural Networks (ANNs), both being able to achieve an AUC higher than 0.9 for common HLA alleles. NetMHCpan better predicts novel HLA molecules when compared to NetMHC, due to using a broader training set. Binding prediction to rare alleles is not as reliable because there is less data. The models are continuously retrained with the event of more experimental data being generated, thus fluctuations in performance are observed. These and other HLA class I binding prediction algorithms are benchmarked weekly on epitopes from the Immune Epitope Database (IEDB) [84] and made available online [85]. Also benchmarked on IEBD is NetMHCcons [86], a consensus algorithm using NetMHC and NetMHCpan. The authors of NetMHCcons concluded that their method yields the highest performance when certain conditions are met, e.g., the allele in question being included in the training, otherwise NetMHCpan is the best predictor. NetMHCpan is used in the aforementioned PGV-001 pipeline, due to its extensive coverage of patient alleles.

Regarding HLA class II binding prediction algorithms, the most accurate with an AUC of 0.875 averaged over 37 sequenced alleles [87] is NetMHCIIpan [87, 88]. Depending on the cell type, class II molecules might not be expressed, making binding prediction superfluous. Consequently, most studies focus on class I epitopes.

The mentioned algorithms are capable of predicting $IC50$ binding affinity, i.e., the concentration of peptide binders where the binding is reduced by half. In general, thresholds of $\log(IC50) < 500nM$ and $\log(IC50) < 50nM$ denote weak and strong binders, respectively [21], where M stands for molar concentration, $mol/L$. Nevertheless, as shown by Paul et al. [89] (where HLA class I $IC50$ cutoffs are

listed), different HLA alleles bind at different $\log(IC50)$ values. Hence, the recommended practice is a rank-based cutoff, where the top 1% of class I peptides and top 10% of class II peptides, based on the predicted $IC50$, are considered binders to a given HLA allele [90].

### 2.6.2 Predicting Post-HLA Binding Events

Higher binding affinity of a peptide to the HLA increases the probability to activate an immune response [91]. However, it is frequently observed that only some of the peptides with good affinity are naturally processed to elicit a response [92]. This can be due to low stability of the peptide-HLA complex. The longer a peptide is bound to an HLA molecule, the more likely it is that the complex leads to the activation of T cells responsible for killing cancerous cells [93]. This stability can be predicted using NetMHCstab [94], which on its own makes moderately good predictions (AUC of 0.86), but in combination with HLA binding predictions, performs slightly better than the individual tools in the predictions [21]. Another theory for the mentioned lack of immune response, is the weak ability of the TCR to bind the peptide-HLA complex. Taking this into account, a learning algorithm [95] that showed prediction of immunogenicity to an extent (AUC of 0.61) was developed but it left ample room for improvement. Lastly, NetTepi [96], combines HLA class I binding affinity predictions with predictions of stability and ability of the TCR to bind the peptide-HLA complex.

### 2.6.3 Prioritizing Neoantigens

Some tumors have a large number of somatic variants and consequently give rise to many potential neoantigens. In these cases it is relevant to rank the large numbers of epitopes to prioritize those that are more likely to elicit an immune response. Without prioritization, the daunting task of screening all epitopes in the laboratory would have to be realized.

An optimal method to model this phenomenon still does not exist. The standard procedure is to sort potential neoepitopes by predicted HLA-binding affinity and possibly also sort by the frequency of HLA alleles in the population as predictive performance for common alleles is generally higher than that for rare alleles [97]. Also, the expression level of the gene is an important factor but it is not sufficient to predict a protein's ability to generate an immune response [98].

In addition, there are public repositories of reported neoantigens, the largest being TANTIGEN [99], SYFPEITHI [100], and the database of T cell-defined tumor antigens [101]. These are good resources for the exploration of known tumor T cell antigens.

## 2.7 Information Visualization

The field of information visualization is based on the idea that visually representing data facilitates users to apprehend raw data values. This translates to more easily finding patterns or identifying unexpected results (e.g., outliers or errors) in the data, when compared to analysing raw data files. In our context, an example of these data files are the VCF files that contain the predicted variants. These files typically contain thousands of rows and multiple verbose columns, consequently making it unpractical to analyse them directly.

A commonly used, freely available, general purpose visualization tool is the Integrative Genomics Viewer (IGV) [102]. It serves as a tool to manually review aligned reads, allowing for confirmation of the variants called [103], thus helping in reducing the occurrence of false positives. This process of visually inspecting variant calls, improves the ability to tune the parameters pertaining to the variant calling software by understanding how it behaves. Besides its main purpose of visualizing the SNVs and SVs called, IGV supports features for the identification of sequencing and analysis artifacts, that may lead to errant SNVs calls, as well as large-scale SVs. The typical workflow is simple: the user loads the reference genome, the resulting BAM file from read alignment and the VCF file with the variant calls to IGV, and then visualizes the called variants. In our context of creating a therapeutic vaccine, discarding false positive variant calls avoids predicting peptides for them. In conclusion, IGV's purpose is two-fold: remove false-positive variant calls and use the knowledge gained to better tune the callers. Figure 2.7 shows the visualization of a SNV using IGV.

## 2.8 Related Pipelines

With the successes achieved using cancer immunotherapy, there is an interest in neoepitope prediction, which lead to the development of pipelines that accomplish just that, e.g., MuPeXi [104], pVAC-Seq [105], PGV-001, and Epi-Seq [106]. In order to give a sense of the general flow of these pipelines, we analyze MuPeXI: it is available as a web service (with a maximum of 6 HLA alleles per submission) that allows to submit a job and get the results by email if desired, and as a portable (unrestricted) application. The inputs to this pipeline are the somatic variant calls in a VCF file, patient HLA types, and optionally a gene expression profile derived from RNA sequencing. MuPeXI returns a sorted list of tumor-specific peptides by priority score, intended to predict the peptides' immunogenicity. This score is dependent on HLA binding affinity of mutant and normal peptides, gene expression level, and allele frequency.

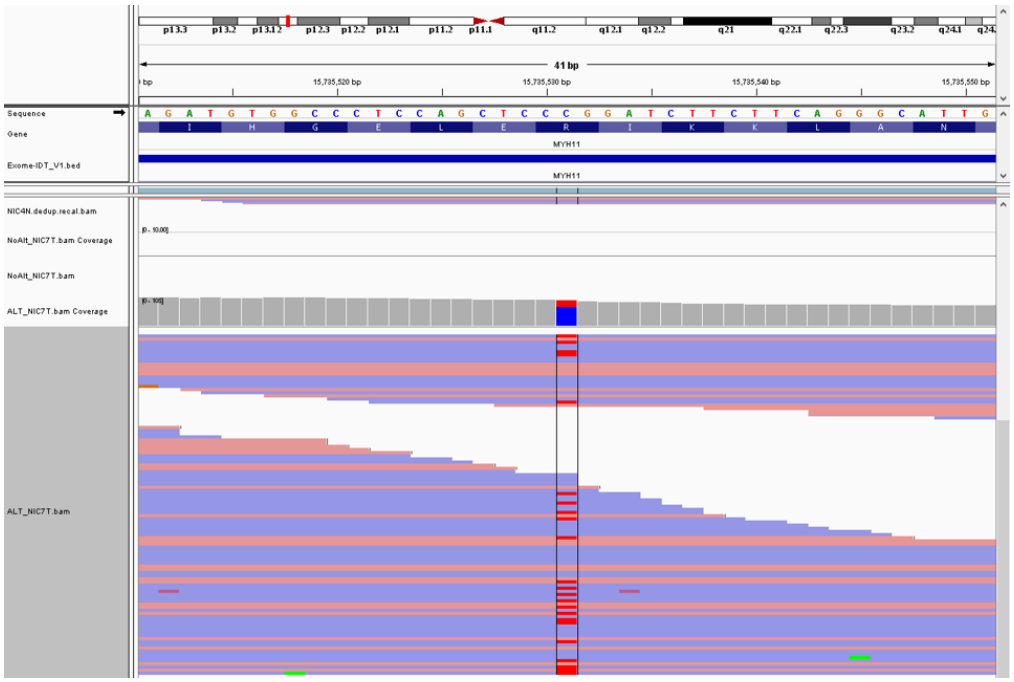


**Figure 2.7:** IGV variant visualisation example. The tumor reads are represented by the many horizontal light blue and light red lines in the bottom part of the picture. In some of the reads we see the variant in bright red, the color code for a T in the software. The non-variant nucleotide was a C, color coded as blue. The proportion of Ts and Cs in the reads at the variant position is given by the colored vertical bar in the middle of the figure. Colored bright green, we can see what could be A artifacts.

## 2.9 Previously used Pipeline at LUMC

Before the development of this work, the pipeline that was used at Leiden University Medical Center (LUMC) consisted of a single makefile script. Makefiles are usually used by the *make* utility [107] to automatically determine which pieces of a large program need to be recompiled and then issue the commands to recompile them. This can be generalized to describe any task where some files must be updated automatically from others whenever these change. In its simplest form, a makefile consists of *rules* in the following form:

**Listing 2.3:** Makefile rule example

```
1 target : prerequisites
2          recipe
```

A *target* is usually the name of a file that is generated by a program and depends on several files. Examples of targets are executable or object files, or an action to execute, such as *clean* to delete files. A *prerequisite* is a file that is used as input to create the target. A *recipe* is an action that *make* carries out and may be comprised of several commands. All in all, a rule explains how and when to remake

certain files, based on the prerequisites, which are the targets of that particular rule. If a prerequisite gets updated, *make* understands that the rule will have to be rerun. The rules present on the previous pipeline makefile can be grouped to correspond to the typical neoantigen detection pipeline steps mentioned in this chapter.

Makefiles scripts are powerful and descriptive for knowledgeable users. The syntax is compact but it has a steep learning curve for the uninitiated, and in general makefile scripts do not favor readability. This makes them harder to be maintained in a collaborative environment. Furthermore, unlike a scripting language such as WDL (described in Section 3.3), there is no well integrated WFMS – as Cromwell (covered in Section 3.4) is for WDL – developed to handle job scheduling and parallelization, two important aspects to leverage the computing capabilities of an HPC cluster. Specifically at LUMC's cluster (detailed in 3.2), running complex makefiles is not supported. Finally, there is also not as finer and as automated control to restart the pipeline's execution from a certain failed point as in WDL with Cromwell. All these factors made it so the previous pipeline could not execute in the LUMC cluster, taking it days to complete running in single and dual CPU socket servers.

# 3

# Methods: Developed Neoantigen Identification Pipeline

## Contents

## 3.1 Overview

This chapter concerns the architecture, context and implementation details of the bioinformatics pipeline developed at and for the Cancer Immunogenomics research group of the Pathology department at Leiden University Medical Center (LUMC). This group specializes in handling large genomics data derived from Next-Generation Sequencing (NGS) technologies. Furthermore, at LUMC there is an HPC Cluster called the Shark cluster [108] (detailed in Section 3.2) where users can submit computational jobs that go into a queue. In this work, the idea was to build an *integrated pipeline*, i.e., a pipeline comprised of several modules that runs seamlessly after issuing a simple command from a Command-Line Interface (CLI). A *module* represents a part of the neoantigen detection workflow mentioned in the last chapter, such as the Somatic Variant Calling module. The pipeline consists of six modules: the first three modules of the pipeline are integrated, whilst the remaining three are not. As such, the latter have to be run separately, not benefiting from the Cromwell Workflow Management System (WFMS) that is covered in section 3.4.

The pipeline's main goal is to identify neoantigens starting from tumor-normal matched tissue samples as illustrated in Figure 3.1. Succinctly, before running the pipeline, tumor-normal matched tissue samples are collected and sent to a sequencing facility that sends back the files with the reads pertaining to the samples. These reads are some of the pipeline's inputs. Afterwards, while the user is connected to the Shark HPC cluster (where the Cromwell WFMS is already setup), the pipeline is executed by running its main script, written in WDL. This triggers Cromwell which is responsible for job scheduling. Cromwell decides – based on the computing resources required and the dependencies between tasks – and submits jobs to the cluster that correspond to modules' tasks. The first three modules of the pipeline – QC and Adapter Trimming, Read Mapping, and Somatic Variant Calling – pull container images from the Docker Hub [109] or Quay [110] container registries, and then build Singularity container images. Pulling and building are handled by the Singularity container tool [111]. Singularity is used due to the safety concerns of HPC clusters. The remaining three pipeline modules – Mutant Protein Prediction, HLA Typing and HLA Binding Prediction – are run separately as they are not yet implemented using WDL. Note that, although this overview is given in the context of the Shark cluster, other HPC clusters (possibly running other backend execution engines), or simpler execution environments (e.g., laptops) can run the integrated pipeline provided they have Cromwell and Singularity adequately configured (detailed in Section 3.7.1). In the coming sections, each step of the pipeline and the development aspects surrounding them are detailed.

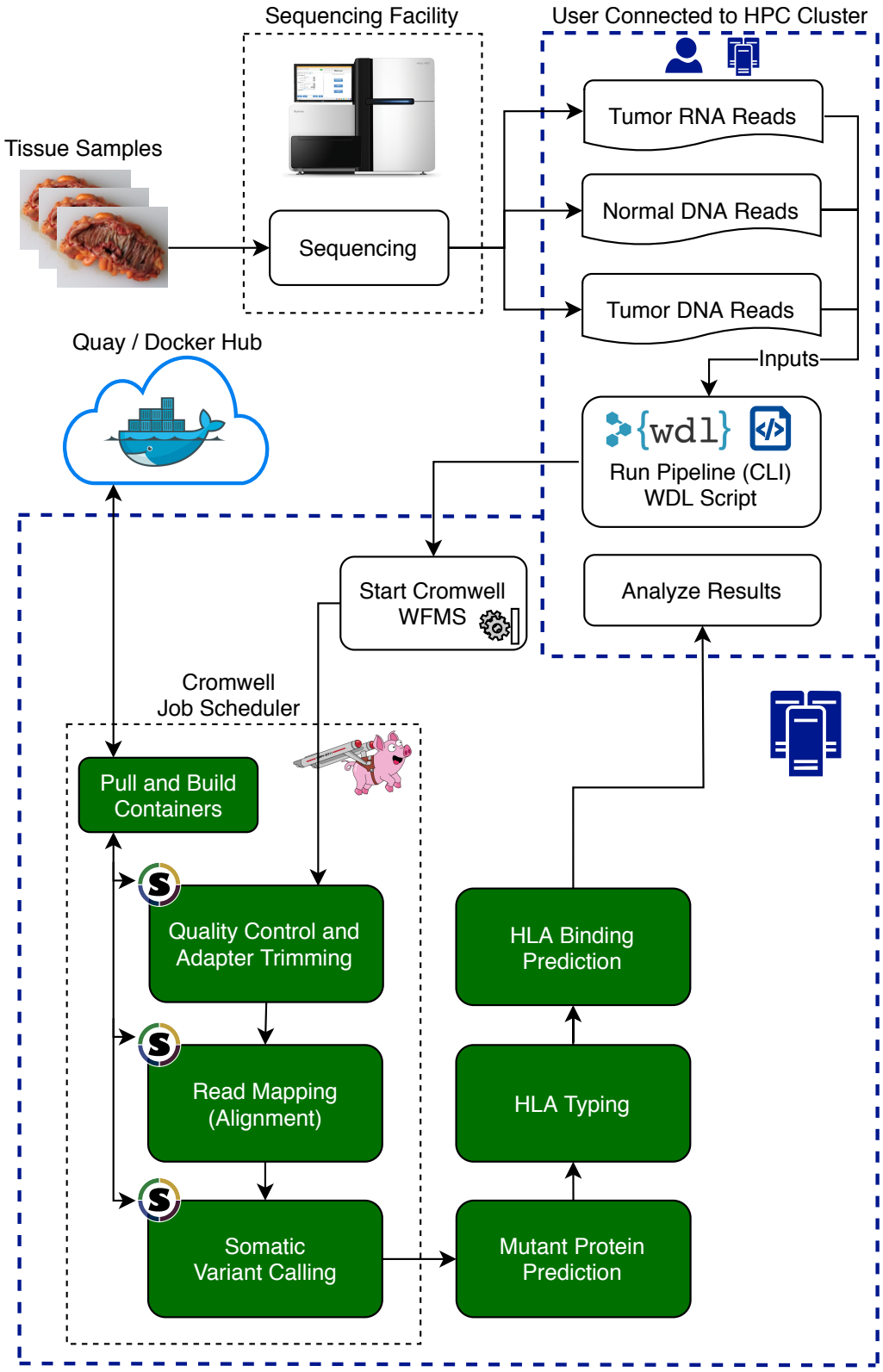


**Figure 3.1:** Neoantigen identification pipeline in the context of LUMC's Cancer Immunogenomics research group which has access to an HPC cluster. The pipeline's steps are highlighted in green, and inside the blue dotted line is what concerns the HPC cluster environment where the pipeline is run.

## 3.2 Shark High-Performance Computing Cluster

The Shark cluster is an HPC cluster located at LUMC that runs with the Open Grid Scheduler/Grid Engine (OGS/GE) [112], a free and open-source batch-queuing system for distributed resource management based on the Sun Grid Engine (SGE). In terms of computing power it has 51 execution nodes totaling 800 CPU cores and 714 TB of raw storage. This is where the pipeline was tested and where it will continue to be run, but its implementation is not bound to the Shark cluster.

The Shark cluster is set up to analyze NGS data coming from Illumina NGS technologies, and optimized to run some of the software mentioned in Chapter 2 – BWA, Bowtie and Tophat for read mapping, and Varscan for variant calling. However, it is not clear how this optimization was accomplished, nor if it also affects software that is run using containers. Besides bioinformatics specific software, the cluster also natively supports common execution and development tools with different versions available: Python [113], R [114], OpenJDK Java Development Kit [115], Oracle's Java Running Environment (JRE) [116], and Git [117]. Moreover, the users can install software that provides virtual environments, for example, anaconda [118]. Consequently, the users can have multiple development environments with different package versions installed.

## 3.3 Workflow Description Language

The pipeline was implemented using WDL, which allows to specify data processing workflows. It is not uncommon to see the word *workflow* used interchangeably with pipeline, as both can refer to the entire sequence of processes used to accomplish some goal. With WDL, one can most notably: define *tasks*, chain them together into workflows, and parallelize their execution (exemplified in sections 3.3.1 and 3.3.2). As users and organizations started using WDL, it became a community driven standard [119]. WDL strives to achieve portability across execution platforms like HPC clusters or cloud platforms. For example, the DNAnexus [120] cloud platform, by using the dxWDL compiler [121], transforms WDL pipelines into an equivalent workflow interpretable by the platform. Moreover, WDL was designed to be accessible and easily understandable by most people that might be interested in building their own workflow, not only by programmers.

WDL workflows are typically run using Cromwell, a WFMS geared towards scientific workflows whose development is linked to the WDL and requires the JRE to run (further detailed in Section 3.4). WDL can be used to write workflows that transparently use container technologies such as Docker, and is suited for describing large-scale workflows in HPC clusters and cloud environments where tasks are scheduled in parallel across numerous nodes. Unlike *makefiles* (discussed in Section 2.9), WDL's syntax is verbose, mostly due to inputs and outputs having to be explicitly defined in a separate JavaScript Object Notation (JSON) [122] file, whereas makefiles make widespread use of regular expressions for the same purpose

– since WDL also supports regular expressions it could be made to work in an equivalent way to makefile but that would take the portability away from it while maintaining its inherent verbosity. Similar to WDL, there is the Common Workflow Language (CWL) [123] whose main goal is portability. As such, CWL workflows are written in the JSON or YAML Ain't Markup Language (YAML) [124] formats (also focused in being portable).

The decision to use WDL stemmed from the fact that already before the start of this work, the Sequencing Analysis Support Core (SASC) team at LUMC had WDL pipeline modules available for sequencing data analysis in their GitHub repository, BioWDL [125], under the MIT license [126]. At the time it already included an in-development somatic variant calling workflow, as well as templates for modular pipelines and a collection of reusable WDL tasks. Additionally, SASC also developed a pipeline development framework, Biopet (Bio Pipeline Execution Toolkit) [127], which is used in some pipeline tasks. The development of the pipeline benefited from SASC's expertise in developing WDL pipelines, and is hosted in the BioWDL repository where it is maintained and updated with newer tool versions and workflows to keep up with the needs of the Immunogenomics group and other users of the pipeline. Since the repository is public, anyone can contribute by making pull requests with improvements.

To explain how workflows and tasks are implemented using WDL, we will now go through a rather simplified example based on the pipeline's Somatic Variant Calling module source code. Starting with workflows and then presenting tasks, we will be highlighting some of the most useful features of the language with examples to motivate them. The code excerpts show line numbers that continue to increment between listings if the code being presented belongs to the same file.

### 3.3.1 Workflows

Having a modular approach to code structure, a WDL script starts by importing the needed dependencies. The imported files are WDL scripts, where other workflows and tasks are defined. Imports can have an alias as shown in Listing 3.1.

**Listing 3.1:** WDL imports

```
1 # File: mutect2.wdl
2 import "tasks/biopet/biopet.wdl" as biopet
3 import "tasks/gatk.wdl" as gatk
```

Then comes the workflow or task definition. In Listing 3.2 the workflow "Mutect" is declared. Its definition starts with the declaration of variables. These variables are *inputs* of type *File*, *String* and *Map*. The File type is practical as it takes file paths and abstracts them into File objects. One useful feature is marking inputs as optional by following the variable type with a question mark, e.g., the "File?

regions" variable declaration. It gives users the option to specify a file comprised of regions in the genome where variants should be called, avoiding unnecessary computation. Another useful feature is assigning a default value to a variable, as seen in the "dockerImages" Map. The latter enables the developer to set a tested container image version as the default, alleviating user burden of input definition and benefiting reproducibility. Input values are specified in a separate file explained in Section 3.7, with an example in Listing 3.10.

**Listing 3.2:** WDL workflow inputs

```
3  workflow Mutect {
4      input {
5          File referenceFasta
6          String tumorSample
7          File tumorBam
8          String? controlSample
9          File? controlBam
10         File? regions
11
12         Map[String, String] dockerImages = {
13           "gatk4":"quay.io/biocontainers/gatk4:4.1.2.0--1",
14           "biopet-scatterregions":"quay.io/biocontainers/biopet-
                 scatterregions:0.2--0"
15         }
16     }
```

After the workflow inputs, we call tasks (usually defined in the imported files). Note that workflows can call other workflows similarly to calling tasks. For the sake of simplicity we will only consider task calls. Tasks also define inputs (detailed in section 3.3.2). When tasks are called, some input values are expected to be passed on by the caller. Similarly to imports, task calls can have an alias. In Listing 3.3, the "ScatterRegions" task, defined in "biopet" (alias for the "tasks/biopet/biopet.wdl" import), gets called under the "scatterList" alias. In this same listing, we see that the task call takes three inputs – "referenceFasta", "regions" and "dockerImage" – that are assigned with values from the workflow inputs (seen in Listing 3.2).

**Listing 3.3:** WDL workflow task calling

```
17     call biopet.ScatterRegions as scatterList {
18         input:
```

```
19              referenceFasta = referenceFasta,
20              regions = regions,
21              dockerImage = dockerImages["biopet-scatterregions"]
22      }
```

WDL has a simple to use built-in mechanism for scattering, i.e., dividing a task's workload into smaller pieces for parallel processing. This allows for multiple smaller jobs to be scheduled simultaneously by Cromwell instead of a single, more computationally demanding one, thus reducing the total execution time of the pipeline. Listing 3.4 displays a task call wrapped within a scatter clause that uses scatter pieces ("scatterList.scatters") calculated in the previous task call (Listing 3.3). Succinctly, the previous task call (aliased "scatterList") produces genome region pieces, "scatterList.scatters", over which the following task calls (aliased as "mutect") execute in a parallel manner.

**Listing 3.4:** WDL workflow scattering a task call

```
23      scatter (bed in scatterList.scatters) {
24          call gatk.MuTect as mutect {
25              input:
26                  inputBams = select_all([tumorBam, controlBam]),
27                  referenceFasta = referenceFasta,
28                  outputVcf = basename(bed) + ".vcf.gz",
29                  tumorSample = tumorSample,
30                  normalSample = controlSample,
31                  intervals = [bed],
32                  dockerImage = dockerImages["gatk4"]
33          }
34      }
```

Finally, the workflow's output is defined, a VCF file with the variants predicted by Mutect (see Listing 3.5). Only the files declared as output in the main pipeline script get saved to the pipeline's output directory. The remaining files are stored in an execution directory used for caching.

**Listing 3.5:** WDL workflow outputs

```
35      output {
36          File outputVcf = mutect.vcfFile
37      }
38  }
```

### 3.3.2 Tasks

Now we will be making the link between the workflow from before and the last task called there, “MuTect”, imported from the “tasks/gatk.wdl” file. Identical to workflows, the first part of a task are the inputs. In Listing 3.6 we highlight two aspects: 1) the correspondence to the inputs assigned in Listing 3.4 when this task is called. 2) The “memory”, “memoryMultiplier” and “dockerImage” inputs which are related to the job execution environment.

**Listing 3.6:** WDL task inputs

```
# File: tasks/gatk.wdl
task MuTect {
    input {
        Array[File]+ inputBams
        File referenceFasta
        String outputVcf
        String tumorSample
        String? normalSample
        Array[File]+ intervals

        Int memory = 4
        Float memoryMultiplier = 3
        String dockerImage = "quay.io/biocontainers/gatk4:4.1.2.0--1"
    }
```

The *command* section in Listing 3.7 contains the commands that are executed to accomplish the task. In this case, the commands are run in an Unix-based container.

**Listing 3.7:** WDL task command section

```
    command {
        set -e
        mkdir -p $(dirname ~{outputVcf})
        gatk --java-options -Xmx~{memory}G \
        Mutect \
        -R ~{referenceFasta} \
```

```
20          -I ~{sep=" -I " inputBams} \
21          -tumor ~{tumorSample} \
22          ~{"-normal " + normalSample} \
23          -O ~{outputVcf} \
24          -L ~{sep=" -L " intervals}
25      }
```

Finally, in Listing 3.8 we define the outputs (similarly to workflows), and the *runtime*. The "vcfFile" output in this listing, becomes available to the workflow that called this task (see Listings 3.2-3.5). There, this output is referred to as "mutect.vcfFile" (Listing 3.5) because the task call was aliased as "mutect" (Listing 3.4). Concerning the runtime section, it is used to specify the computing resources that should be allocated when the job is submitted to the cluster, and the environment where the commands are run. In Listing 3.8, using input variables, the amount of (primary) memory and the container image to be used are specified.

**Listing 3.8:** WDL task output and runtime sections

```
26      output {
27          File vcfFile = outputVcf
28      }
29
30      runtime {
31          docker: dockerImage
32          memory: ceil(memory * memoryMultiplier)
33      }
34  }
```

### 3.3.3 Structs

In the type of analysis that our pipeline does, it is not uncommon that files need to be indexed, thus increasing the number of files having to be referred to. However, there is a WDL construct that allows to group files with their indices: structs (exemplified in Listing 3.9). Structs are not limited to grouping files and their indices, they can group many and other types of variables. Structs can be declared in a separate WDL file that is then imported or in files containing tasks, workflows, or both.

**Listing 3.9:** WDL struct grouping a file and its indices

```
26 struct BwaIndex {
27     File fastaFile
28     Array[File] indexFiles
29 }
```

## 3.4 Cromwell WFMS

Cromwell is a WFMS that runs workflows written in CWL or WDL (used in this work). It is an open source project under the BSD 3-Clause license [128], therefore private and commercial use is permitted. Having this license contributes to the adoption of the pipeline by others as there are no licensing costs. Cromwell is geared towards scientific workflows and its development is linked to WDL. Furthermore, it manages job submission and scheduling without pipeline developers having to write code for that. However, the environment where the pipeline is run needs to have Cromwell configured for it first. This is trivial if users are to run the pipeline in a laptop, but not as straightforward in an HPC environment (see Appendix A.1).

Some of Cromwell's features are: simplicity in leveraging container technology (see Listing 3.8); support for popular backends – Apache Spark [129], Google Cloud [130], SGE [19] and others; caching that avoids re-running tasks previously executed. Figure 3.2 outlines how caching works in Cromwell. When the pipeline gets run, the database connected to the Cromwell installation in the system – where all workflow runs are logged to – is queried for information concerning the jobs about to be scheduled. If the same command with the same user-defined inputs has already been run, Cromwell inspects the file system for the presence of the output files that the previous identical jobs generated. In case the files exist, depending on Cromwell's configuration, either links pointing to them or copies are created in the current execution folder.

Cromwell can run in one of two modes: "run" or "server". Run mode serves as a way to get acquainted and experiment with Cromwell, or run very simple workflows as only one workflow can run simultaneously using the same command. For more complex workflows server mode is recommended, as it exposes a larger set of features through its Representational State Transfer (REST) endpoints. To customize Cromwell's behaviour and to configure it to run with a given backend in server mode (SGE in our case) a configuration file written in Human-Optimized Config Object Notation (HOCON) is used. HOCON's primary goal is to keep the semantics from JSON (tree structure, set of types, encoding and escaping), while being more convenient as a human-editable configuration file format [131]. Cromwell's HOCON configuration file contains configuration values related to the endpoint, and backend parameters related to job submission, runtime resources, file system, caching, database and more. In Appendix A.1 the configuration file used in Shark cluster is presented, detailing the the level of customization possible.

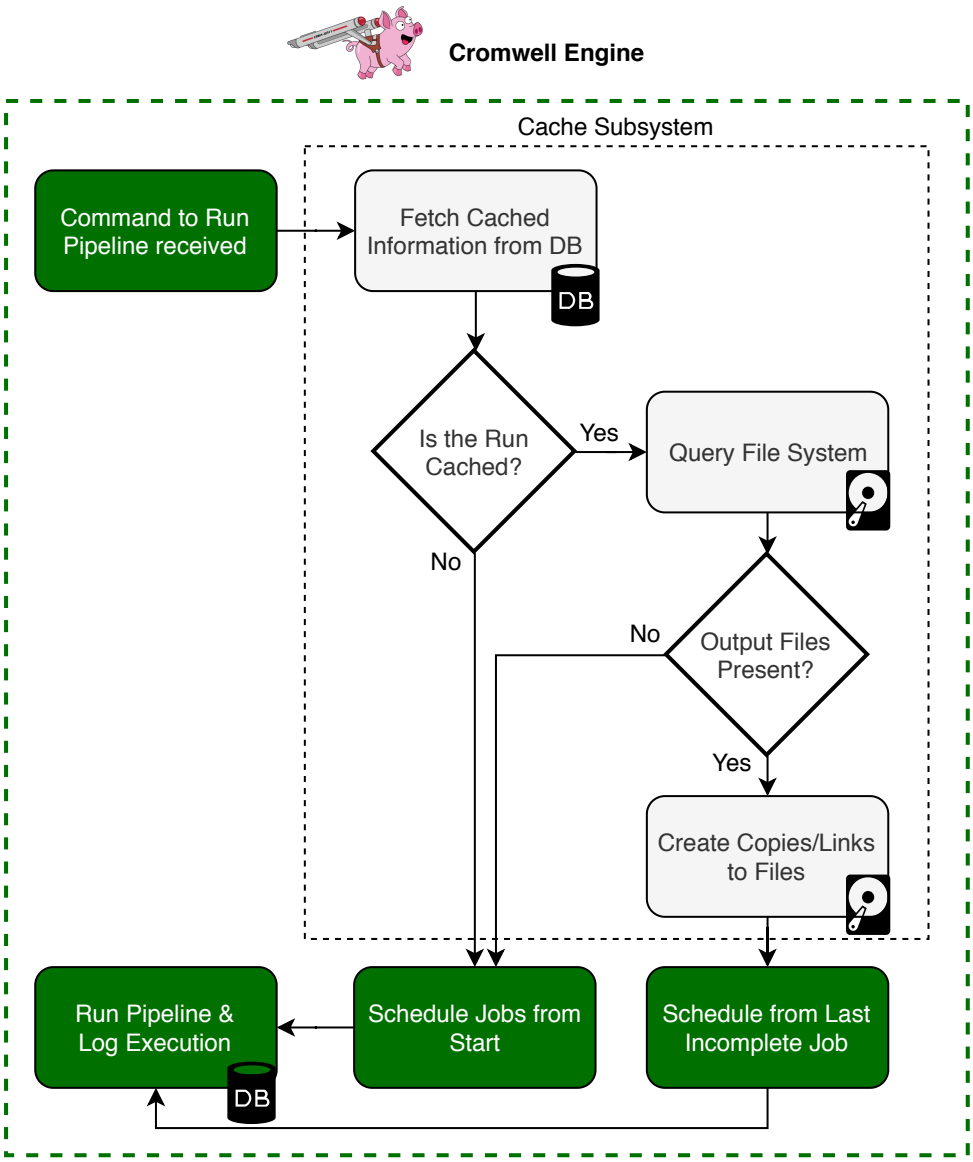


**Figure 3.2:** Cromwell cache subsystem: when a pipeline is run, the database associated to the Cromwell server running in the system is checked for pre-existing information about the jobs about to be submitted. In case output files already exist from previous runs, either links pointing to them or copies are created in the current execution folder.

## 3.5 Singularity Container Platform

Containers are a great tool to simplify software deployment and execution. In the context of our pipeline, containers address the important aspect of building a reproducible and portable pipeline. Containers allow different users to have the same software configuration by simply linking to the same Docker image (see Listing 3.6). This results in having the same configured software versions even if the users are using different environments, thus providing consistent results between environments. Furthermore, time is saved due to avoiding installing software. This benefit is amplified when we consider that a given user can have multiple, or changing over time, computing environments. The container image only needs to be downloaded once. Generally, when software is updated, simply updating the image URL pointed to by the pipeline is enough. The latter can be done in the user inputs passed on to the pipeline (detailed in Section 3.7), avoiding changing the source code. This ease in selecting software versions also makes it easier to experiment with different versions. The requirement to run containers is having containerization software installed.

Singularity was the chosen containerization tool because it is the one installed in the Shark cluster. Compared to the more popular Docker, Singularity has a stricter security model, hence its availability in the cluster. Nevertheless, Singularity allows not only to build the containers available in the Singularity Container Library [132], but also those in other container platforms' repositories, for example, in Docker Hub or in Red Hat's Quay repositories. Since the latter two repositories are more popular, many bioinformatics tools are already available there. In our case, all the required bioinformatics tools were available in these repositories, mostly in the Biocontainers organization repository in Quay [133]. This means that we did not have to containerize any software, presenting another advantage of using containers in this work.

## 3.6 Integrated Pipeline Code Structure

The integrated pipeline continued being developed and is now in release 4.0.0 [18]. Our contribution to it finished just before version 1.0.0 [17] was launched. The integrated pipeline's source code, as seen in the respective GitHub repository at the last commit pertaining to this work [134], is divided into the following git submodules [135]:

- "QC": responsible for adapter trimming and gathering QC metrics using FastQC.
- "BamMetrics": gathers metrics from BAM files using Picard.
- "gatk-preprocess": produces a report on the observed base quality scores and optionally recalibrates the BAM file based on the report.

- "somatic-variantcalling": responsible for running part, or all the variant callers (user-defined) – Mutect2, VarDict and Strelka – and combining the resulting VCF files.
- "tasks": collection of reusable tasks called by other workflows. It includes, for example, read mapping tasks.

Essentially, Git submodules are Git repositories that are kept as subdirectories of another Git repository. The division into git submodules enables integrating them easily into other WDL pipelines. Taking into account WDL's architecture of workflows and tasks, the submodules accomplish a good degree of code decoupling, i.e., low dependency between their code. All submodules are dependent on the "tasks" submodule. Otherwise, they are mostly independent and code changes should not involve rewriting multiple parts of the pipeline, and the parts that would need alterations should be easily traceable. Located at the root of the repository we find the main script, "pipeline.wdl" which runs the entire pipeline (see how in Section 3.7).

## 3.7 Running the Integrated Pipeline

### 3.7.1 Environment Configuration

Although the pipeline was mainly developed and tested in the Shark cluster – running the free and open-source Debian Linux distribution Ubuntu 14.04.6 LTS (kernel version "3.13.0-170-generic x86_64") – the pipeline should run in most Unix-based Operating Systems (OSs). The major requirement is that the execution environment, be it a laptop or a computing cluster, should have Cromwell installed and properly configured (covered in Section 3.4). With that ensured, Cromwell can interpret WDL or CWL scripts regardless of the Unix-based OS where it is installed. Contributing to platform independence is that the pipeline modules' software runs within containers managed by Singularity, which is available for Unix-based OSs. Neither a wide range of OSs nor multiple tool versions were validated, but in principle and succinctly, the pipeline should run on most current Unix-based OSs installed with Java 8 and above, Singularity 3, and Cromwell 43 to 45.

### 3.7.2 Pipeline Inputs

#### 3.7.2.A Inputs Definition File

Before running the integrated pipeline, the user must define the inputs in a JSON file. Listing 3.10 shows an example excerpt of an input file used to run the pipeline. In that example, paths for the output (results) directory and for the reference genome are specified, as well as, configuration and computing resource parameters concerning the Strelka and VarDict variant callers, respectively. In pipelines with multiple

workflows and tasks (such as ours), the input file can get rather verbose. A more complete example can be seen in Appendix A.2.1. Such dense input files are not user-friendly and can be error-prone. Currently, there is no alias mechanism (similar to the one seen with workflows and tasks) that could avoid this complexity. Only the definition of appropriate default values suiting most cases reduces input definition complexity (by avoiding having to define those default variables).

**Listing 3.10:** Pipeline's user-defined input file example excerpt

```
{
  "pipeline.outputDir": "/storage/output",
  "pipeline.reference": {
    "fasta": "/ref/Homo_sapiens_assembly38.fasta",
    "fai": "/ref/Homo_sapiens_assembly38.fasta.fai",
    "dict": "/ref/Homo_sapiens_assembly38.dict"
  },
  "pipeline.somaticVariantcalling.SomaticVariantcalling.strelka.
     Strelka.strelkaSomatic.exome": true,
  "pipeline.somaticVariantcalling.SomaticVariantcalling.vardict.
     VarDict.varDict.memory": 16,
  "pipeline.sampleConfigFile": "/config/samples.yml"
}
```

#### 3.7.2.B Sample Configuration File

The last input in Listing 3.10 is defined when the users want their samples aligned – the users do not have to necessarily align their samples if they have already done it, they could instead choose to start with the Somatic Variant Calling module. Listing 3.11 is an example of a simple sample configuration file corresponding to the last input definition of Listing 3.10. In it, we see that identifiers are given to each sample ("TUMOR" and "NORMAL"), library ("lib1" for both samples) and readgroup ("lane1" for both libraries). Additionally, the directories where the R1 and R2 reads are located are defined. More samples, libraries and lanes can be defined, it simply depends on the data the user wants processed by the pipeline (see Appendix A.2.2 for a more complex example). Also in Listing 3.11, note that the "TUMOR" sample defines the "NORMAL" sample as being the control. This is the case of tumor-normal matched samples. The correspondence of samples, libraries and readgroups to the pipeline's workflow is made clearer in Section 3.8.2.

**Listing 3.11:** Pipeline's YAML sample configuration file example

```
samples:
  - id: TUMOR
    control: NORMAL
    libraries:
      - id: lib1
        readgroups:
          - id: lane1
            reads:
              R1: /storage/fastq/TUMOR_L003_R1.fastq.gz
              R2: /storage/fastq/TUMOR_L003_R2.fastq.gz
  - id: NORMAL
    libraries:
      - id: lib1
        readgroups:
          - id: lane1
            reads:
              R1: /storage/fastq/NORMAL_L003_R1.fastq.gz
              R2: /storage/fastq/NORMAL_L003_R2.fastq.gz
```

### 3.7.3 Womtool Input Generation

Womtool is a helpful CLI tool whose main uses are validating WDL scripts (syntactical and semantically), and generating a JSON skeleton file of the inputs available for a given workflow (exemplified in Listing 3.12). Concerning the latter use, the generated JSON includes, for each input, its type, its default value (if any), and whether it is optional. The full output, including optional variables, is about 400 lines long.

**Listing 3.12:** Womtool "inputs" example command, taking the main pipeline script, "pipeline.wdl", as the source workflow. The command is followed by an excerpt of its output, which by default includes optional inputs.

```
$ java -jar womtool-45.jar inputs pipeline.wdl
{
  "pipeline.somaticVariantcalling.SomaticVariantcalling.vardict.
     VarDict.varDict.minimumVariantDepth": "Int (optional, default =
     4)",
  "pipeline.sample.Sample.library.Library.readgroup.Readgroup.qc.
     QC.FastqcRead1.NoneFile": "File? (optional)",
  "pipeline.somaticVariantcalling.SomaticVariantcalling.mutect2.
     Mutect2.gatherVcfs.memory": "Int (optional, default = 8)",
  "pipeline.sample.Sample.library.Library.readgroup.Readgroup.qc.
     QC.Cutadapt.bwa": "Boolean? (optional)",
  "pipeline.somaticVariantcalling.SomaticVariantcalling.strelka.
     Strelka.strelkaSomatic.memory": "Int (optional, default = 4)",
  "pipeline.bwaIndex": {
    "fastaFile": "File",
    "indexFiles": "Array[File]"
  }
}
```

### 3.7.4 Running the Pipeline

The command used to run the pipeline at the Shark cluster, assuming all needed files are in the current directory, is:

**Listing 3.13:** Example Cromwell command, running the pipeline's main WDL workflow

```
cromwell run pipeline.wdl -i inputs.json -o options.json
```

where "pipeline.wdl" is the main pipeline script, "inputs.json" is the file assigning values to the pipeline's WDL inputs (explained in section 3.7.2), and "options.json" is a settings file configuring Cromwell's behaviour, for example, by disabling using cached results. When the command in Listing 3.13 is run, the workflow in "pipeline.wdl" is executed by Cromwell. A detailed log is printed to the CLI as the workflow progresses. The log shows what tasks are being scheduled, the commands they run (that were defined

as in Listing 3.7), and the tasks status. The latter is given by the status of the jobs submitted to the cluster. Figure 3.3 shows an example of a Cromwell's log during a variant calling workflow.

```
[2019-08-07 19:59:07,39] [info] BackgroundConfigAsyncJobExecutionActor [96bbc5faMutect2.mutect2:0:1]: executing: # make sure there is no pre
existing Docker CID file
rm -f /home/amfgcpaulo/repos/somatic-variantcalling/cromwell-executions/Mutect2/96bbc5fa-1c82-497b-994f-cd6b51c35b58/call-mutect2/shard-0/ex
ecution/docker_cid
# run as in the original configuration without --rm flag (will remove later)
docker run \
  --cidfile /home/amfgcpaulo/repos/somatic-variantcalling/cromwell-executions/Mutect2/96bbc5fa-1c82-497b-994f-cd6b51c35b58/call-mutect2/shar
d-0/execution/docker_cid \
  -i \
  --user $EUID \
  --entrypoint /bin/bash \
  -v /home/amfgcpaulo/repos/somatic-variantcalling/cromwell-executions/Mutect2/96bbc5fa-1c82-497b-994f-cd6b51c35b58/call-mutect2/shard-0:/cr
omwell-executions/Mutect2/96bbc5fa-1c82-497b-994f-cd6b51c35b58/call-mutect2/shard-0:delegated \
  quay.io/biocontainers/gatk4@sha256:4c9e4fc6143e4632fd9f4515234481c4824944d0a09d10dff7d2e2e7235c752b /cromwell-executions/Mutect2/96bbc5fa-
1c82-497b-994f-cd6b51c35b58/call-mutect2/shard-0/execution/script

# get the return code (working even if the container was detached)
rc=$(docker wait cat /home/amfgcpaulo/repos/somatic-variantcalling/cromwell-executions/Mutect2/96bbc5fa-1c82-497b-994f-cd6b51c35b58/call-mut
ect2/shard-0/execution/docker_cid)

# remove the container after waiting
docker rm cat /home/amfgcpaulo/repos/somatic-variantcalling/cromwell-executions/Mutect2/96bbc5fa-1c82-497b-994f-cd6b51c35b58/call-mutect2/sh
ard-0/execution/docker_cid

# return exit code
exit $rc
[2019-08-07 19:59:07,64] [info] BackgroundConfigAsyncJobExecutionActor [96bbc5faMutect2.mutect2:0:1]: job id: 15646
[2019-08-07 19:59:07,66] [info] BackgroundConfigAsyncJobExecutionActor [96bbc5faMutect2.mutect2:0:1]: Status change from - to WaitingForRetu
rnCode
```

**Figure 3.3:** Cromwell's log output example.

## 3.8 Integrated Pipeline Modules

In this section, we use Unified Modeling Language (UML) sequence diagrams to show the process flow and interactions between the WDL workflows of the pipeline. We made two adaptations to the diagrams that are outside of the current UML specification [136]: 1) Although not in the sequence diagram specification, we make use of composition arrows (distinguished by the filled in diamond on one end). They convey a multiple simultaneous call to the same workflow or task, serving as an abstraction to the occurrence of input division into smaller pieces. Usually, this means scattering the genome into smaller regions so that each job (run in parallel) takes less time to complete. 2) The parallel combined fragment in Figure 3.7 is divided vertically instead of horizontally to save vertical space.

In the sequence diagrams that follow, lifelines (horizontal rectangles on the top with text and a dashed line descending from them) correspond to WDL workflows that call tasks throughout their execution specifications (vertical grey rectangles). Some data transformation and some optional tasks were omitted to achieve a clearer diagram. With the same goal, arrows returning from workflows show only the most relevant returned outputs, and some workflow and task names were slightly changed when compared to the source code.

### 3.8.1 Quality Control and Adapter Trimming

The sequence diagram in Figure 3.4 shows that the first step pertaining to the QC and Adapter Trimming module is splitting the FASTQ files – format shown before in Listing 2.1 – resulting from sequencing the tumor and normal tissue samples. These files can be multiple GB in size, containing millions of text lines. By splitting them into smaller files, the "QC" workflow jobs seen in Figure 3.4 can be called in parallel. Following the guidelines in section 2.2.2, FastQC is run twice, before and after trimming adapter sequences with Cutadapt. Finally, by running the MultiQC [137] tool, FastQC reports are aggregated and plotted into a single HTML file.

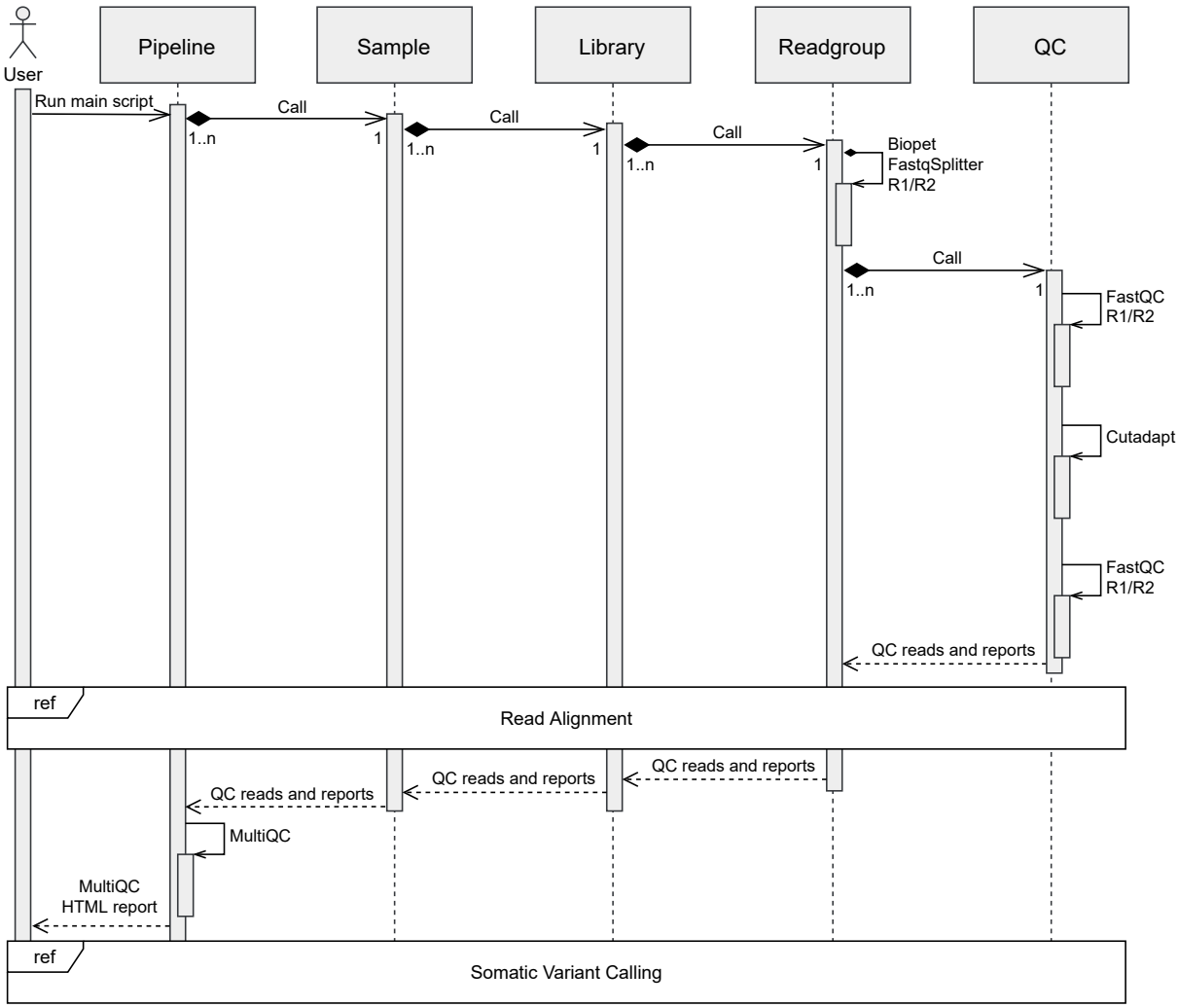


**Figure 3.4:** QC and Adapter trimming WDL workflow represented in a UML sequence diagram.

Figure 3.5 shows the "Mean Quality Scores" plot in MultiQC, generated from FastQC data. This figure showcases MultiQC's sample highlighting (lines in blue). MultiQC also allows filtering out samples from the plots, zooming in on any given part of the plots, exporting the plots to different formats and

saving the applied viewing options. In Figure 3.5, we see that there is a left side bar with links to not only other FastQC plots (bottom), but also to plots presenting data from other tools used in the pipeline – Picard, SAMtools and GATK – as MultiQC automatically gathers all reports found in the specified folder (and its subfolders). Having all reports in a single file that allows filtering is practical and helps to ensure sample quality and to spot incorrect pipeline execution. More plots generated by MultiQC can be seen in in Appendix B.

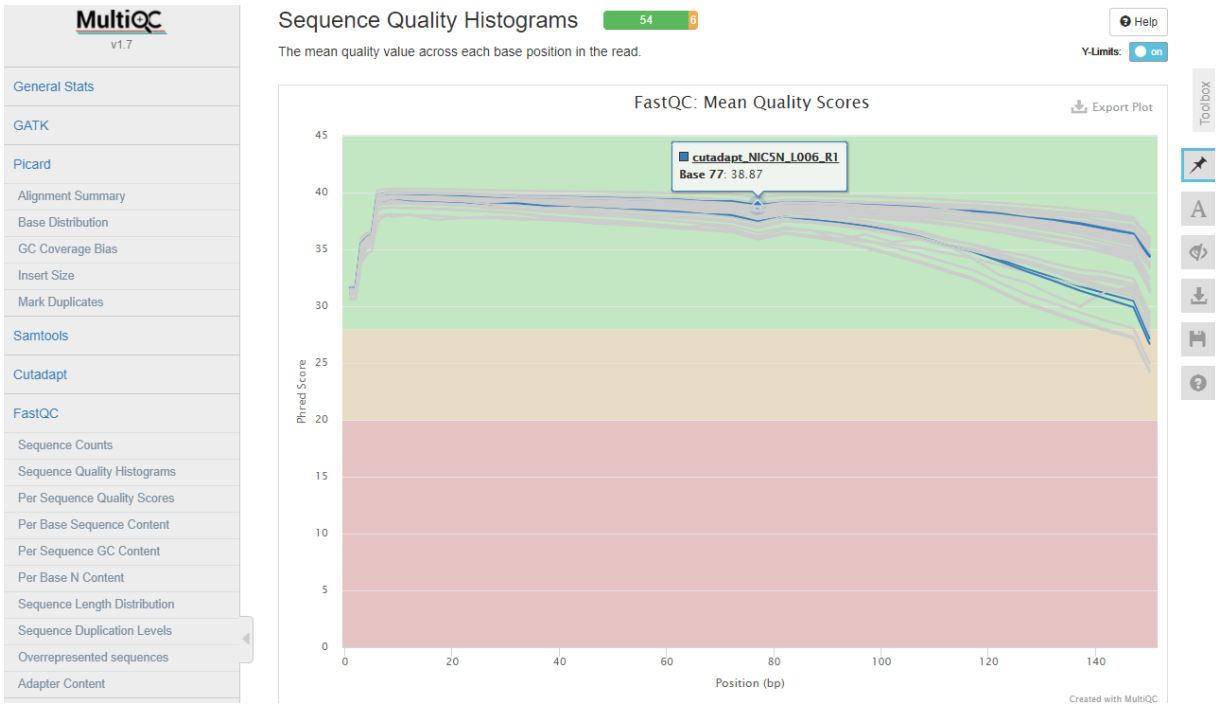


**Figure 3.5:** MultiQC's "FastQC: Mean Quality Scores" plot with samples highlighted in blue.

### 3.8.2 Read Mapping (Alignment)

Reads are mapped using BWA-MEM [138], the BWA tool used to align reads ranging from 70 base pairs (bps) to a few megabases (ours being over 150 bps). BWA-MEM was chosen over Bowtie 2 (both mentioned in section 2.2.3) for three main reasons: 1) generally, BWA-MEM has a better sensitivity [139]. 2) GATK best practices for alignment [140] use BWA-MEM. We follow the updated version of the protocol [141]. 3) Since BWA-MEM was already used in the previous pipeline, it allowed us to then make a fairer comparison between the two pipelines. A comparison between the variants called (which are influenced by the alignment step) is presented in Section 4.2.3. It was paramount to the Immunogenomics group to have at least the same variants being called by both the new and the previous pipelines.

Figure 3.6 summarizes the Read Mapping module: after the scattered FASTQ files (from sequencing)

are processed in the QC and Adapter Trimming module, they are aligned against a reference genome (defined by the user), resulting in BAM files. Then, a Picard task is called, "MarkDuplicates", that locates and tags *duplicate reads* (originating from a single DNA fragment) in the BAM files. Afterwards, calling tasks that run GATK tools, the BAMs are recalibrated: tables for Base Quality Score Recalibration (BQSR) are generated over each scattered genome region (created by the "ScaterRegions" task) and then gathered into one table (with the "GatherBqsrReports" task). Subsequently, BQSR is applied to the BAMs files and they are gathered into a single BAM (per "Library" call) with Picard. As a post-processing step, the "BamMetrics" workflow is run, collecting various metrics from the recalibrated BAM file that will later be collected by MultiQC (mentioned in Section 3.8.1). Finally, the "Sample" workflow receives the BAM and metrics files for each "Library" workflow that it initially called. The BAM files are merged into one (using SAMtools) for each sample specified in the YAML sample configuration file (exemplified in Listing 3.11).

### 3.8.3 Somatic Variant Calling

Depending on the user-defined inputs (covered in Section 3.7.2.A), the Somatic Variant Calling module runs up to three different variant callers: Mutect2, Strelka2 and VarDict. Optionally, the user can choose to combine the variant callers outputs into a single VCF file. The logic behind running all three variant callers is to maximize the number of true protein changing variants, be it a SNV, indel, or SV. Not all neoantigen detection pipelines do this, e.g., Epi-Seq focuses only on SNVs. Table 3.1 summarizes the types of variants called by each of the variant callers used.

**Table 3.1:** Variant types called by Mutect2, Strelka2 and VarDict. It is also specified whether the variant caller reports SNVs that are located together, i.e., MNVs.

| | **SNV** | **MNV** | **Indel** | **SV** | **Germline** |
|---|---|---|---|---|---|
| **Mutect2** | Yes | Yes | Yes | No | No |
| **Strelka2** | Yes | No | Yes | Yes | Yes |
| **VarDict** | Yes | Yes | Yes | Yes | Yes |

The variant calling tools were chosen according to four main considerations mentioned in Section 2.3. Firstly, Mutect and Strelka's showing higher sensitivity at low VAF for WES data with lower false positive rates than other tools. Secondly, VarDict achieving high sensitivity with a carefully chosen VAF while calling all types of variants. Thirdly, the inputs required being BAM aligned reads of matched tumor-normal samples. Finally, combining different variant callers resulting in more variants called. Additionally, Mutect and Strelka were already used in the previous pipeline.

Figure 3.7 summarizes the Somatic Variant Calling module: with the matched tumor-normal BAM files from Read Mapping, the three variant calling workflows – "VarDict", "Mutect2" and "Strelka2" – are called in parallel. In each workflow, a scatter task is first executed, dividing the genome into multiple

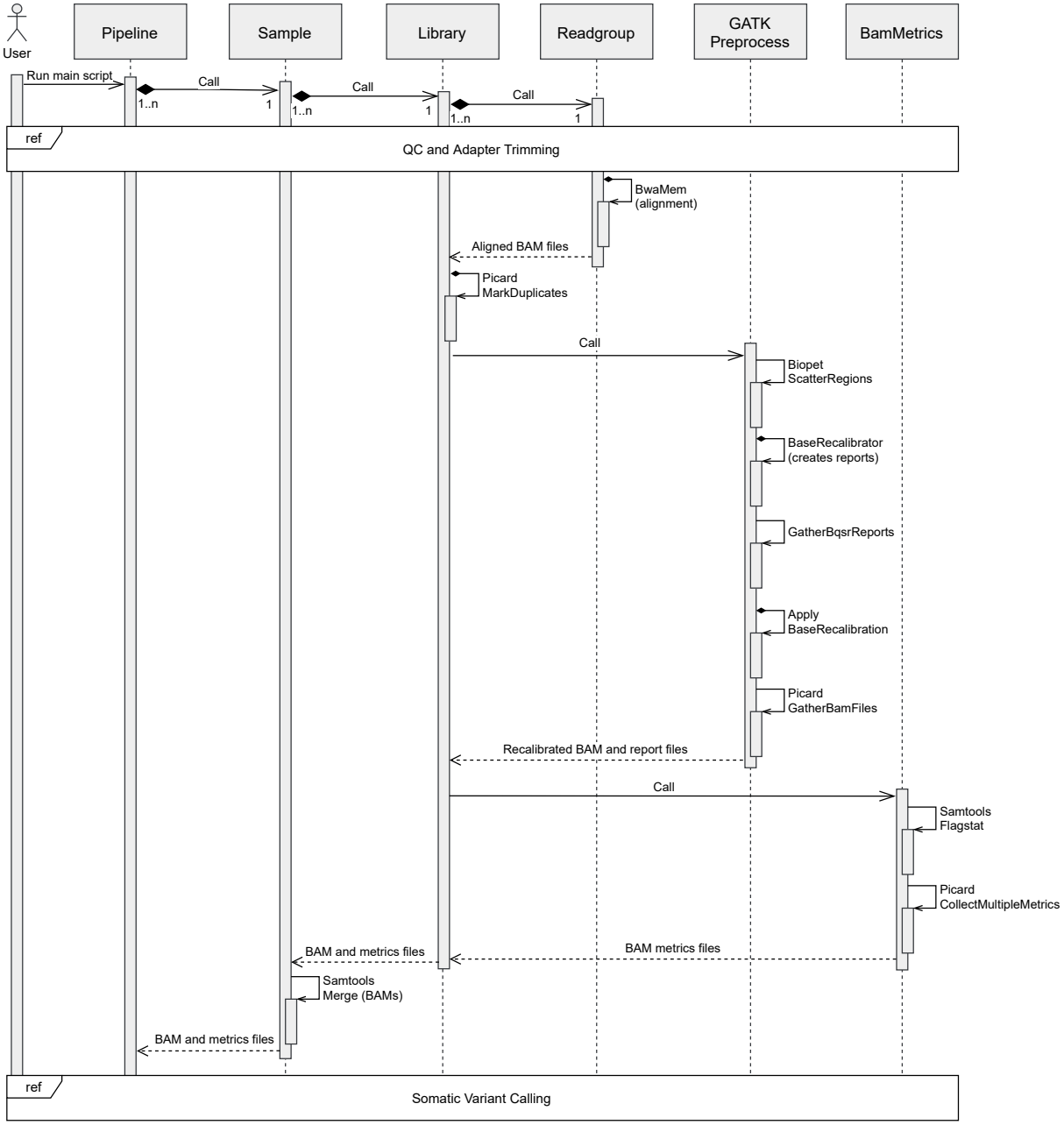


**Figure 3.6:** Read Mapping (Alignment) WDL workflow represented in a UML sequence diagram.

regions. Consequently, each workflow can call its variant callings tasks – "VarDict", "MantaSomatic", "StrelkaSomatic" and "Mutect2" – in parallel (conveyed by the black-filled diamonds in the diagram). After the three variant calling workflows are over, the resulting VCF files are returned and combined (provided the user defined the respective pipeline input variable). Since each variant caller workflow runs differently, we will analyse each one as well as address the combination of their outputs.

#### 3.8.3.A Mutect2

The first task called by the "Mutect2" workflow scatters the genome regions into smaller pieces over which the "Mutect2" tasks are called in parallel. Each of these tasks generates a stats file that needs to be merged into one – done with the "MergeStats" task – for later use in the variant filtering step. There are two noteworthy workflows (now in the true meaning of the word, not in the WDL sense) that the "Mutect2" tasks follow: the Read Orientation Artifacts workflow and the Panel of Normals workflow, both part of the GATK best practices for calling somatic mutations using Mutect2 [142]. The integration of these two workflows is part of the improved filtering contribution mentioned in Section 1.3.

The Read Orientation Artifacts workflow is advised for samples that are suspected to exhibit orientation bias artifacts. This applies to all FFPE tumor samples and samples sequenced on Illumina platforms such as the HiSeq platform that we used [143]. Nevertheless, with the current optimizations this workflow can be run without affecting result accuracy nor causing considerable extra computing [142]. This workflow requires passing the "–f1r2-tar-gz" parameter to Mutect2, resulting in a compressed file with counts related to read orientation. This file is then used in the "LearnReadOrientationModel" task that outputs a model of the read orientation artifacts – a bias detected by its distinct signature [144]. Then, "GetPileupSummaries" summarizes read support for a set number of known variant sites into a table. These known variants, and other useful resources, can be obtained from the GATK resource bundle [145–147]. Using the output from the previous task, the "CalculateContamination" task is run. The latter calculates the fraction of reads coming from cross-sample contamination. Finally, and after merging the scattered VCF files from Mutect2's variant calling, the variant calls can be filtered – "FilterMutectCalls" task – using the outputs from the "LearnReadOrientationModel" and "CalculateContamination" tasks.

The Panel of Normals workflow first requires creating a database using the variants found in the normal tissue of various patients. Then, by passing the database path to Mutect2's variant calling command, we get improved variant filtering as there is data on variants that we know were found in the tissue of normal patients. The logic is that, if a candidate somatic variant was detected in one patient but it was present in the panel of normals database, then it is likely to be a normal variant and can thus be filtered.

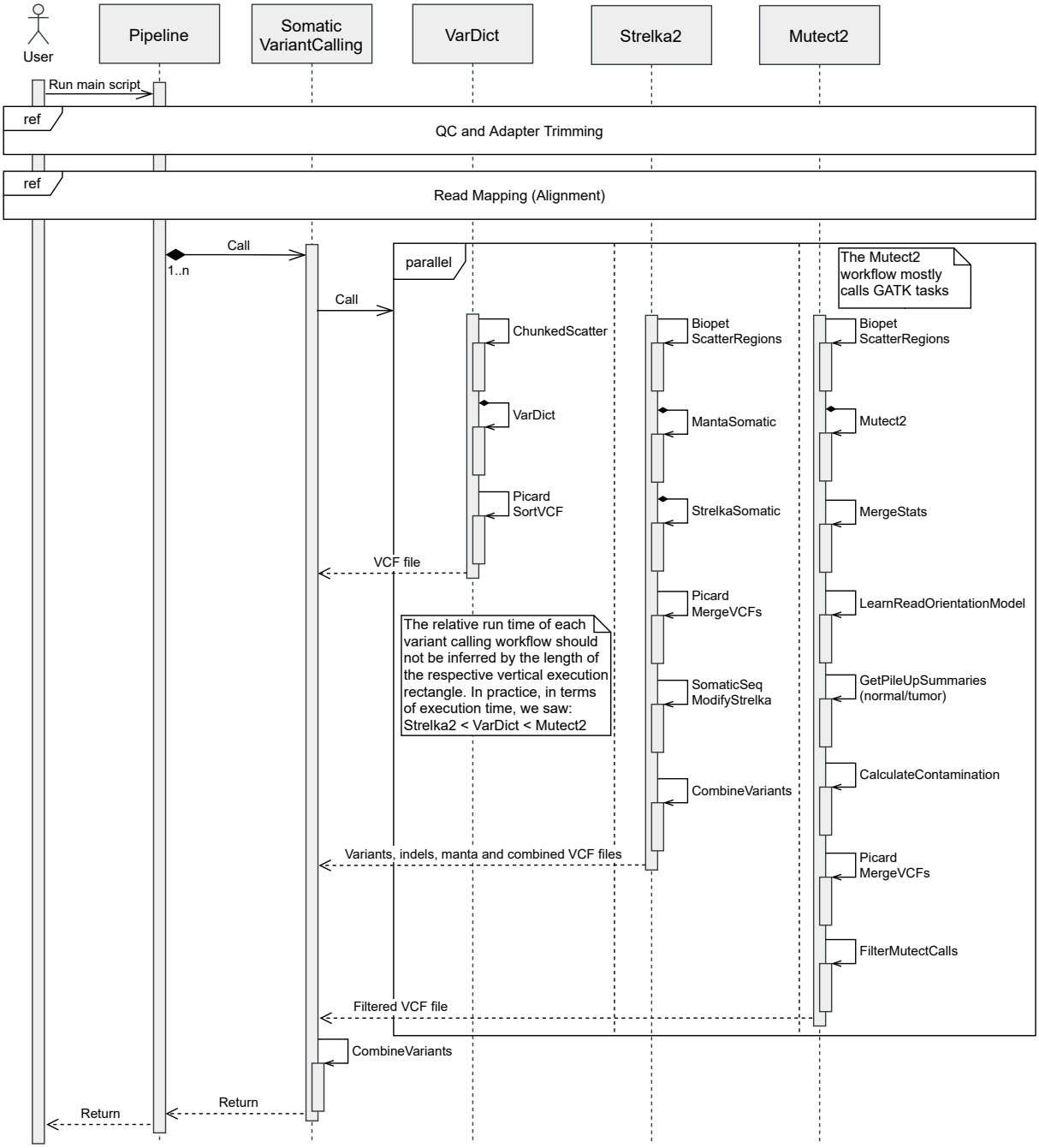


**Figure 3.7:** Somatic Variant Calling WDL workflow represented in a UML sequence diagram.

#### 3.8.3.B Strelka2

Similarly to Mutect2, the first task called by the "Strelka2" workflow scatters the genome regions into smaller pieces. Over which the "MantaSomatic" – SV calling – and "StrelkaSomatic" – SNV and indel calling – tasks are called in parallel. Identically to Mutect2, the scattered VCF files for each type of variant called are merged with Picard's MergeVFCs tool. However, since with Strelka2 different tools are used to generate each type of variant, the VCFs can be combined in the "CombineVariants" task. For this, we first need to run an auxiliary script from SomaticSeq to adequately format the VCFs.

#### 3.8.3.C VarDict

As opposed to the other two variant callers, VarDict is limited to processing regions of up to 1 Mbps. This lead to the development of the chunked-scatter tool [148]. Briefly, compared to Biopet's scatter-regions, chunked-scatter provides a finer control over the division of the genome regions. The mentioned VarDict limitation and how chunked-scatter tackles it is detailed in Section 4.1.3.

After having the genome regions chunked, multiple VarDict tasks run in parallel, one per chunk. VarDict's fastest implementation in terms of execution time is written in Java [149]. The VarDict WDL tasks can be made to run in less time by increasing the "threads" input available to the pipeline users. VarDict is configured following the software's recommendations for paired sample variant calling from BAM files. After running the main VarDict Java binary, R and Perl scripts are used for variant filtering. The main binary's output is piped directly into the filtering scripts.

Concerning filtering and as mentioned in Section 2.3, the VAF threshold is of special relevance in VarDict. We found that with the default VAF threshold of 2%, the true variants called were in line with the results from the other two variant callers (detailed in Section 4.2). However, as many other parameters, the user can define another value. Furthermore, we decided to run variant filtering with the "-A" flag, thus outputting all predicted variants at the same position (instead of choosing only one), and with the "-M" flag, increasing the stringency in the filtering of candidate somatic variants.

Finally, the chunked VCFs must be gathered into one file. VarDict's VCFs require sorting, so Picard's SortVCF is used (also responsible for merging).

#### 3.8.3.D Combining Variant Calls

Last in the Somatic Variant Calling module, the returned VCFs from each variant caller are (optionally) combined into a single VCF file for more practical analysis in downstream tools. With this, the integrated pipeline returns to the main script and finishes, leaving the user a directory with all the results, except intermediary files (e.g, scattered regions) which are kept in the cromwell execution folder for caching.

## 3.9 Standalone Pipeline Modules

In this section, we cover the pipeline modules that are not integrated into the Cromwell pipeline. This means they are run separately.

### 3.9.1 Mutant Protein Prediction

In the Mutant Protein Prediction module, Isovar is run with RNA-Seq data extracted from the tumor sample (covered in Section 2.4). It uses RNA to assemble the most abundant coding sequence for each mutation. Isovar provides a Python API and can be used as a CLI tool. It works by (adapted from its documentation [150]):

1. Collecting RNA reads which span the location of a variant,
2. Filtering the RNA reads to those which support the mutation,
3. Assembling mutant reads into longer coding sequences,
4. Matching mutant coding sequences against reference annotated reading frames, and
5. Translating coding sequences determined directly from RNA into mutant protein sequences

Figure 3.8 presents an overview of Isovar and highlights one of the advantages of using RNA to determine the coding sequence for mutations: the phasing of somatic and germline variants. If phasing is not taken into account, the resulting protein sequences may not match the ones produced by tumor cells, thus being an incorrect choice to include in a tumor vaccine. In case the RNA-Seq reads from a patient do not contain enough information for Isovar to predict the mutant protein, data from the Ensembl VEP is used (mentioned in Section 2.4). Initially, we intended to modify Isovar's code to, for example, output more than only the top predicted protein sequence. However, Isovar restarted active development during the same period of our work [151], becoming less of a priority.

### 3.9.2 HLA Typing

The HLA Typing module remained the same when compared to the previous pipeline. OptiType, as mentioned in Section 2.5, is the most accurate for HLA class I. OptiType takes the sequencing reads from a given patient (FASTQ files) and maps them against all HLA class I alleles, from which a binary matrix is generated indicating which alleles a specific read could be aligned to with the least number of mismatches [77]. Based on this matrix, an optimization problem – formulated as an Integer Linear Program (ILP) – is solved, having as output the predicted optimal HLA alleles.

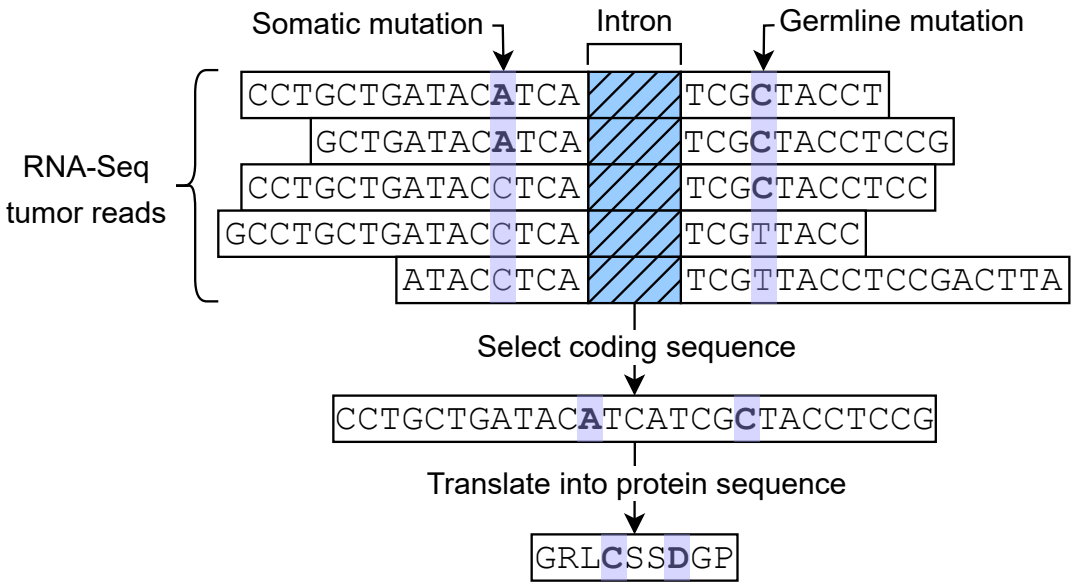


**Figure 3.8:** Isovar tool overview, showing the phasing of somatic and germline mutations. Note that RNA-Seq uses a cDNA library, hence the presence of T instead of U. Adapted from [65].

### 3.9.3 HLA Binding Prediction

For the HLA binding prediction module, a Python tool was developed [152] – licensed under GNU General Public License version 3 [153]. It depends on a modified version of mhctools [154], a Python Application Programming Interface (API) to commonly used MHC binding predictors. We use mhctools to run NetMHC and NetMHCpan (mentioned in Section 2.6.1 as performing the best in HLA class I binding prediction). Our modified version of mhctools [155] enables functionality that exists in NetMHCpan but that is not made available by mhctools. This modification enables mhctools to output the different types of peptide ranking that NetMHCpan allows, that depend on the data used to train this predictor [156] (detailed in Section 3.9.3.D).

Our Python tool, henceforth referred to as bind-pred, receives as inputs:

- a file containing the predicted mutant protein sequences from Isovar (see Section 3.9.1), alongside the non-mutant sequences (example excerpt in Listing 3.14);
- size values, to extract short peptides around the mutation locations of the given sizes;
- a protein database (explained below);
- the patient HLA alleles as predicted by OptiType (see Section 3.9.2) that we want to predict the short peptides binding to;
- names of the output directory and to prefix the name of the created files.

As outputs, bind-pred writes:

- A tab-separated file comprised of concatenated data output by netMHC and netMHCpan (see Section 3.9.3.D);
- Files for use in mass spectometry and peptide reactivity testing (see Section 3.9.3.E);
- A time-stamped log file with debug, general, warning and error data.

The protein database we choose as input is Swiss-Prot, a high quality manually annotated and non-redundant protein sequence database from UniProt Knowledgebase [157]. Additionally, we filtered the database to only contain human protein sequences, leaving the database with around 20 thousand sequences. Swiss-Prot was choosen for 2 main reasons: fast query time and avoiding wrongly annotated proteins (common in non-curated databases). The purpose of using a database is explained in the description of bind-pred that follows.

The first step in bind-pred is dividing the (mutant and non-mutant) proteins in the input file into short peptides of user-defined sizes around the mutation positions. When we run bind-pred, we typically choose peptide sizes of 8 to 12 because HLA class I molecules preferentially bind short peptides [158]. The second step is querying the database for complete matches of the short mutant peptides. If any of the latter are found in the database, they are filtered out in a third step. The fourth and last step is calling netMHC and netMHCpan. We will now describe each step in more detail.

**Listing 3.14:** Bind-pred protein input file example. It contains shorter proteins than usual for increased legibility (usually 51 amino-acid long, not 25 as shown).

```
1 varID                WT                          MUT
2 chr6_113860425_C/T   AAEEPSKVEEKKAEEAGASAAACEA   AAEEPSKVEEKKAAEAGASAAACEA
3 chr12_20654151_G/A   AGLMPGKWVEDSDESGDTDDPEEEE   AGLMPGKWVEDSDTSGDTDDPEEEE
```

### 3.9.3.A Dividing Input Proteins into Short Peptides surrounding Mutations

In the first step of bind-pred, the mutation positions are located. Since we do not know the number of mutations nor how they are organized, and because the two protein sequences – Mutant (MUT) and non-mutant (called Wild Type (WT)) – may differ in size, we start by aligning them against each other through global pairwise alignment. To perform this alignment, we used the Biopython [159] package. In pairwise alignment, a gap function that dictates the penalty in opening and extending gaps is required – ours is presented in Listing 3.15. To obtain better alignments, we more heavily penalized opening gaps to the right of the sequence, i.e., if there is to be a gap, we want it to come sooner rather than

later. Additionally, we had to take into account that the two protein sequences can be of different sizes, making the gap function depend on the longest one.

**Listing 3.15:** Gap function used in aligning the input protein sequences (WT and MUT)

```
# 'x' is the gap position in the sequence and 'y' is gap length.
def gap_function(x, y):
    if y == 0:  # no gap
        return 0
    elif y == 1:  # gap open penalty
        return -0.6 + (x/len_longest_prot) / 2
    return -0.6 + (x/len_longest_prot) / 2 - 0.1 # gap extension penalty
```

Listing 3.16 shows an alignment between two protein sequences as it is printed to the log of bind-pred. In this listing, we see that there was a deletion that caused the MUT sequence to have 2 gaps in the alignment.

**Listing 3.16:** Alignment of two protein sequences printed to the log of bind-pred. On the top is the WT, and on the bottom the MUT. A 6 nucleotide deletion occurred (indel variant), leading to a 2 amino-acid gap in the middle of the MUT sequence.

```
KIHVTPLIPGKSQSVSVSGPGPGPGPGLCPGPNVLLNQQNPPAPQPQHLAR--
||||||||||||||||||||||||  ||||||||||||||||||||||||
KIHVTPLIPGKSQSVSVSGPGPGPG--LCPGPNVLLNQQNPPAPQPQHLARRP
```

With the mutation locations stored, we then break the proteins into smaller peptides around those locations (according to the user-defined sizes). We generate all the possible peptides of the defined sizes containing the mutations. This is done using a sliding-window over a big enough chunk surrounding the mutation. Besides the files mentioned in Section 3.9.3.E, a file with all the short peptides (with identifiers) is created.

#### 3.9.3.B Querying Protein Database for Exact Peptide Matches

In the second step, we query the user-defined database (of known proteins) with the short peptides from the file generated in the previous step. The querying is done using an improved implementation of the Basic Local Alignment Search Tool (BLAST) algorithm [160], BLAST+ [161], providing rapid sequence comparison. In this step, we again use the Biopython package, that provides an interface to BLAST CLI tools. More specifically, we run the "blastp-short" tool, as we are querying for short peptides. This

results in an XML file with the query data containing values such as the number of amino-acid matches between the query sequence and the one found in the database (which does not need to completely match).

#### 3.9.3.C Filtering Exact Matches

In the third step, we filter out the short peptide sequences (from step 1) which completely align with no mismatches to the sequences in the database. This is done by parsing, with a Biopython method, the XML file from step 2.

#### 3.9.3.D Running Binding Prediction Software

In the last step, we run netMHC and netMHCpan to predict the binding of the short peptides that remained after filtering, to the patient's HLA alleles. We not only predict binding to the MUT peptides, but also to the corresponding WT peptides (helpful in downstream analyses). We call netMHCpan twice, the difference between the two calls being the "Binding Affinity" parameter ("-BA"). With this parameter, peptide ranking changes as the data used to train the ANN is different. To accomplish the two distinct calls to netMHCpan we had to modify mhctools (mentioned in Section 3.9.3). Listing 3.17 shows an example excerpt of the final output (tab-separated) file where the binding predictions are concatenated together.

**Listing 3.17:** Bind-pred binding predictions output file (example excerpt). We show the same 3 peptide sequences for each binding prediction function call – netMHCpan-BA, netMHCpan and netMHC. The "Bind_Level" column indicates the predicted binding strength. Lines 3, 4 and 7 show a weak binding (WB) prediction, line 10 shows a strong binding (SB) prediction, and the remaining lines show a no-binding prediction (N/A).

```
1 Var_id  %Rank   Bind_Level  WT_peptide  MUT_peptide WT_Affinity MUT_Affinity
        DAI HLA_allele  Size    Program
2 chr3_167720076_G/A-L93-WT25-MUT25-M0    70.1093 N/A PLYAVMYP    PLYAVMYS
      43320.6 44286.9 -966.30  HLA-A24:02  8   netMHCpan-BA
3 chr3_167720076_G/A-L93-WT25-MUT25-M1    1.8626  WB  LYAVMYPV    LYAVMYSV
      2698.3  2333.8  364.5   HLA-A24:02  8   netMHCpan-BA
4 chr3_167720076_G/A-L93-WT25-MUT25-M2    1.0265  WB  YAVMYPVF    YAVMYSVF
      643.6   930.9   -287.29 HLA-A24:02  8   netMHCpan-BA
5 chr3_167720076_G/A-L93-WT25-MUT25-M0    65.0    N/A PLYAVMYP    PLYAVMYS
      N/A N/A N/A HLA-A24:02  8   netMHCpan
6 chr3_167720076_G/A-L93-WT25-MUT25-M1    3.6779  N/A LYAVMYPV    LYAVMYSV
```

```
      N/A N/A N/A HLA-A24:02  8   netMHCpan
7 chr3_167720076_G/A-L93-WT25-MUT25-M2    1.8158  WB  YAVMYPVF    YAVMYSVF
      N/A N/A N/A HLA-A24:02  8   netMHCpan
8 chr3_167720076_G/A-L93-WT25-MUT25-M0    70.0    N/A PLYAVMYP    PLYAVMYS
      42322.65    43924.73    -1602.08 HLA-A24:02  8   netMHC
9 chr3_167720076_G/A-L93-WT25-MUT25-M1    3.0 N/A LYAVMYPV    LYAVMYSV
      3760.65 3697.55 63.09   HLA-A24:02  8   netMHC
10 chr3_167720076_G/A-L93-WT25-MUT25-M2   0.25    SB  YAVMYPVF    YAVMYSVF
      108.38  116.21  -7.82  HLA-A24:02  8   netMHC
```

#### 3.9.3.E Outputs

Besides the binding prediction output file above, during the first step of bind-pred (see Section 3.9.3.A), two files are generated while the proteins input file – containing the MUT and corresponding WT sequences – is parsed. One of the files is useful for doing mass spectrometry, which is done in the lab (outside the scope of this work). This file consists of variant identifiers that are generated based on the mutation position with the respective mutant sequence alongside it (see Listing 3.18). The other file is useful for peptide reactivity testing, which is done in the lab (and outside the scope of this work). It contains variant identifiers followed by MUT sequences of size 25 that are built around the respective mutation position (see Listing 3.19).

**Listing 3.18:** Example excerpt of output file for use in mass spectometry

```
1 >chr3_167720076_G/A_M26
2 MTMEEMKNEAETTSMVSMPLYAVMYSVFNELERVNLSAAQTLRAAFIKAEK
3 >chr5_136213548_C/T_M26
4 NDEVNEGELKEIKQDISSLRYELLEKKSQATGELADLIQQLSEKFGKNLNK
5 >chr5_141346161_G/A_M26
6 NDSDLTLYLVVAVAAVSCVFLAFVIMLLALRLRRWHKSRLLQASGGGLAST
```

**Listing 3.19:** Example excerpt of output file for use in peptide reactivity testing

```
1 var_id  MUT_peptide
2 chr3_167720076_G/A  SMVSMPLYAVMYSVFNELERVNLSA
3 chr5_136213548_C/T  QDISSLRYELLEKKSQATGELADLI
4 chr5_141346161_G/A  AAVSCVFLAFVIMLLALRLRRWHKS
```

## 3.10 R Shiny Result Analysis

The only interface to the pipeline mentioned until now is the CLI. However, data visualization was a concern since the conceptualization stage of this pipeline. Taking into account the benefits of data visualization mentioned in Section 2.7, a separate R Shiny [20] application was extended to plot variant data distributions. Shiny is an R package to build interactive web applications with a focus on simplicity of development and redistribution, responsiveness in the interaction with the application for the end-user and appealing default User Interfaces (UIs) based on Bootstrap [162], a widely used front-end component library. The UI can be written in R but also directly using the common web development languages – HTML, CSS and JavaScript.

Our Shiny application's source code is available in GitHub [163] and hosted online [164] (experimental) in the Shinyapps cloud [165]. The contribution to this module was adding a results tab, containing two bar plots, and the respective data scraping script that gathers and processes the data used to build the plots. An overview is shown in Figure 3.9: on the top left, the user can browse directories and is expected to select one with a specific structure (defined by the Immunogenomics group). Then, the data scraping script locates and processes the files containing the variants and the resulting peptides. These variants underwent visual inspection by the research group and have three main attributes used in generating the plots: "selected", "expressed" and "variantTYPE". "Selected" if they alter the amino acid sequence, "expressed" if they are being expressed according to the RNA data, and "variantType" being the variant type. The "Selected" and "variantType" data are obtained using Ensembl VEP.

One plot shows the number of protein changing variants per sample and whether they are expressed (see Figure 3.10). The other plot shows the type of variants (see Figure 3.11). An R script was written [166] to generate the data needed for the plots according to the samples the user selected. This module requires further development so that others outside the Pathology group are able to use it. Nonetheless, it was already proven to be useful, as it helped detecting an error related to the processing of a patient sample (covered in Section 4.1.4).

## 3.11 Continuous Integration and Testing

For CI purposes - the practice of merging in small code changes frequently, rather than merging in a large change at the end of a development cycle - Travis CI [167], a continuous integration platform, is used. Although normally requiring a paid license, it is free for open source projects such as our integrated pipeline. Travis CI automatically builds and tests code changes, providing immediate feedback on the success of the change. The goal of using this tool is to build more reliable software by developing and testing in smaller increments. Additionally, the pytest framework [168] was used to write the integration tests [169]. The pipeline's Travis CI main configuration file [170] calls the pytest framework to run the


**Figure 3.9:** R Shiny module's results tab overview. On the left, the user can browse folders. On the right, the plots generated with the data from the chosen folder are shown. Figures 3.10-3.11 show the plots in more detail.

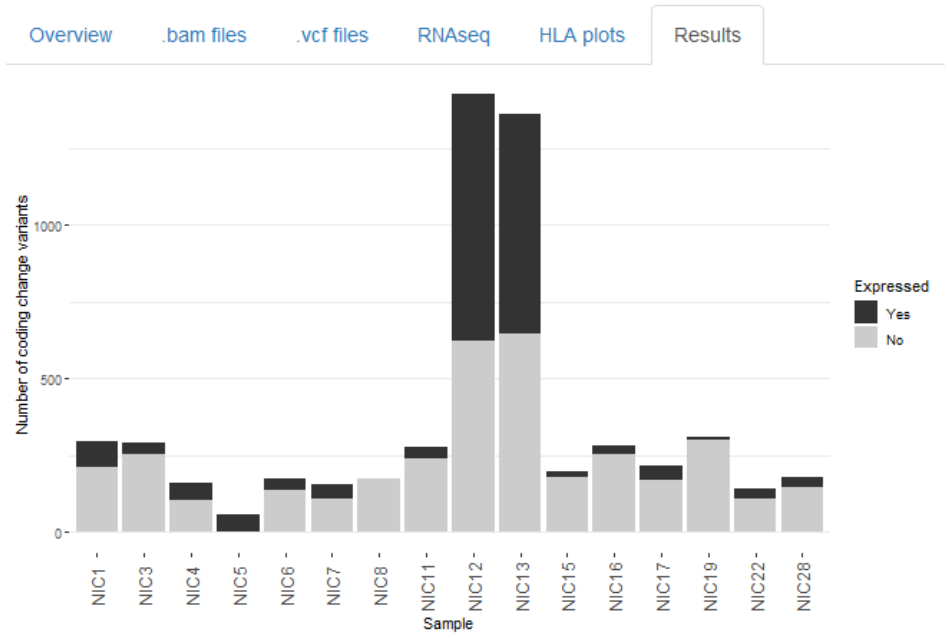


**Figure 3.10:** R Shiny module plot showing the number of protein changing variants per sample, while indicating whether the variants are expressed (black) or not expressed (grey).

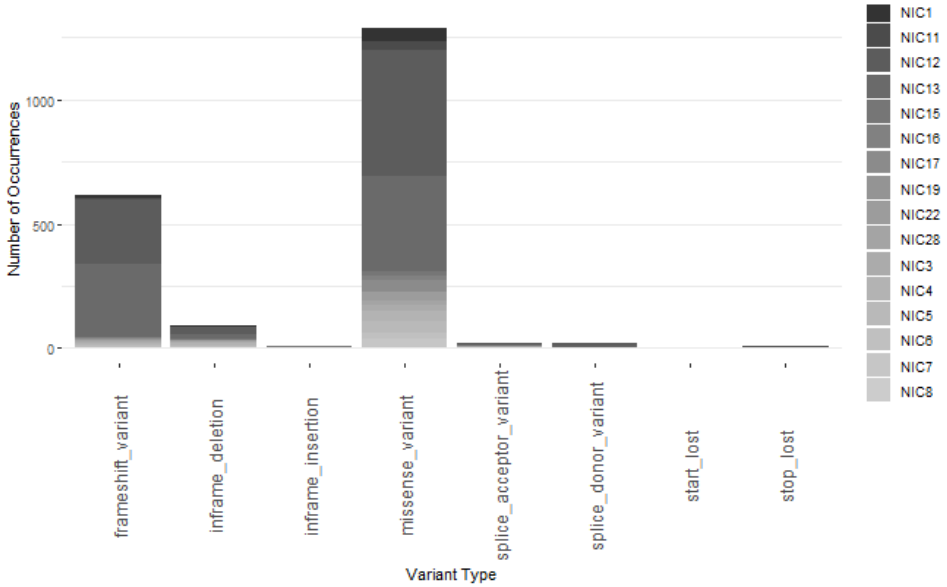


**Figure 3.11:** R Shiny module plot showing the number of occurrences for a set of variant types. The samples' identifiers are shown on the right.

tests. These tests use very small test data: instead of GB of files as is the case when running the pipeline for real-world analysis, files in the order of KB are used, speeding up the CI process. The tests verify that the pipeline runs successfully and whether the outputs change after new code is pushed to the repository. For example, in the Somatic Variant Calling module, inputs smaller than 1 Megabyte containing NGS reads are used to check whether the variants called remain the same with new code.

# 4

# Problems and Validation

## Contents

## 4.1 Problems and Noteworthy Aspects

In this section we go over some of the notable problems and interesting aspects we found while developing the Cromwell pipeline.

### 4.1.1 Cromwell Filling Cluster's Storage Space

When executing the pipeline, Cromwell requires the inputs to be localized following a certain directory structure. To avoid filling up the cluster's storage with duplicate inputs, Cromwell first tries to create hard-links, then soft-links and finally a copy of the file. When using container images, the soft-links strategy is not possible, thus if hard-links fail to be created a file copy ensues, resulting in the storage being filled.

The Network File System (NFS) in the Shark Cluster has a maximum of 1000 hard-links per file. This limit was reached after multiple pipelines sharing the same inputs were run, resulting in the storage quota allocated for the Pathology group being filled. Furthermore, creating copies takes time, which slowed down the pipeline execution considerably.

The solution for this problem was, after a file reached 1000 hard-links, copy it once and then create hard-links over this newer copy. If the 1000 hard-links are reached in the newer copy, repeat the process.

### 4.1.2 BWA Intricacies

During a phase of result comparison between the Cromwell pipeline and the previously used one, we noticed that, for the same sample, there were different variants called as a result of unexpected differences in read alignment. We concluded that the differences in the read alignment were a consequence of not defining in the Cromwell pipeline's inputs a certain index file for BWA. That index file makes BWA alt-aware, i.e., it informs BWA of mapping regions that are alternative, making it prioritize mapping reads to the primary (non-alt) regions. The noteworthy aspect that we realized was that even if the mapping to the alt region is better than to that of the primary region, BWA will preferentially map to the primary region as long as the reads reasonable align to it. This was an unexpected behaviour that changed our perception of the alignment process.

### 4.1.3 VarDict Memory Usage

During testing of the pipeline, we noticed there were VarDict jobs that hung for hours. After logging in to the computing nodes where the VarDict jobs were running, it became clear that the VarDict processes were thrashing due to the Java Virtual Machine (JVM) constantly reaching the maximum virtual memory allowed for the process (even after increasing the virtual memory limit to unreasonable amounts). This caused Java's garbage collector to constantly clear up the processes' virtual memory, impeding the

process to move forward. The thrashing was solvable by giving VarDict genome regions not bigger than 1 Mbps in size to call the variants in, but for this the pipeline's region scattering logic had to be changed. Consequently, and as mentioned in Section 3.8.3.C, the chunked-scatter tool was developed.

To understand how this tools differs from the previously used scatter-regions tool, we first have to understand how the genome regions are divided into a format variant callers recognize. Region segments are written in a BED format file. BED files contain genome segments written line by line. A line is composed by an identifier followed by the start and end coordinates (in bps). A line could have, for example, "chr1" as an identifier (for chromosome 1), and "1 1000000" as coordinates – Listing 4.1 shows an example of a BED file. The root cause for VarDict's thrashing is that it uses memory linear to the individual segments size in the BED files [171], rapidly filling the memory for large contiguous segments in the BED file. Having segments of a maximum of 1 Mbps effectively avoided thrashing.

**Listing 4.1:** Example BED file ("regions.bed") with differently sized genome segments, both for chromosome 1

```
1 $ cat regions.bed
2 chr1    100     2000000
3 chr1    3000000    4500000
```

Listings 4.2 and 4.3 show the different behaviour of the aforementioned tools when taking as input the BED file from Listing 4.1. With chunked-scatter (new strategy), the segments are divided even if they were contiguous in the corresponding input segment which was not done in the scatter-regions tool (old strategy).

**Listing 4.2:** Chunked-scatter genome region division into 1 Mbp chunks

```
1 $ chunked-scatter -c 1000000 regions.bed
2 $ cat scatter-0.bed
3 chr1    100     1000100
4 chr1    999950  2000000
5 chr1    3000000 4500000
```

**Listing 4.3:** Scatter-regions genome region division using 1 Mbp as the scatter parameter (-s).

```
$ scatter-regions -s 1000000 regions.bed
$ cat scatter-0.bed
chr1    100     2000000
chr1    3000000 4500000
```

### 4.1.4 R Shiny Module Error Finding

As alluded to in Section 2.7, visualizing data is a practical way to identify unexpected results. An example of this is given by the "NIC5" sample values in Figure 3.10. In this figure we see that the "NIC5" sample did not contain any non-expressed variants, which was caused by human error in handling the output files. Quickly identifying errors avoids later issues and saves troubleshooting time.

## 4.2 Validation

Pipeline validation was done using two types of data: 1) synthetic data, i.e., data semi-computationally fabricated, meaning that we know what variants were injected into the original data. These variants are gathered in what we refer to as a ground truth file. We followed two approaches to obtain synthetic data: searching for synthetic datasets available online (see Section 4.2.1), and generating our own synthetic datasets using tools with that purpose (see Section 4.2.2). 2) Using real patient data that had already been processed by the previous pipeline, meaning we know which somatic variants were called before. Although we cannot be sure the variants called before represent the ground truth, as they are simply a product of another pipeline with the same objective, our goal is that the newer pipeline calls at least the same protein changing variants as the previous pipeline (see Section 4.2.3). With both types of data, the validation consists of comparing the variants called by our pipeline with the ones in the ground truth VCF files.

### 4.2.1 Validation with ICGC-TCGA DREAM Synthetic Data

As a first validation, we assessed the pipeline's variant calling performance. We ran the Somatic Variant Calling module of our pipeline on synthetic genomes from the ICGC-TCGA DREAM Mutation Calling Challenge that took place in 2013 [172]. A set of three simulated tumors, each paired with its normal sample, were available. The (WGS) reads were aligned to the hg19 reference. Moreover, for each simulated tumor, a VCF file containing the ground truth (computationally added) variants was provided. We chose this dataset because it is commonly used for the assessment of somatic variant callers.

#### 4.2.1.A Methodology

Before succeeding to run the Somatic Variant Calling module, we had to address three aspects regarding VarDict. The first aspect was VarDict failing to call variants using the hg19 reference file as it contained "M" and "R" characters – these mean A or C, and A or G, respectively. Since there were only three occurrences and given that they did not overlap with any variant in the ground truth file, we replaced them with one of the possible nucleotides, in this case A. The second aspect was VarDict only running successfully for one of the three sets, restricting our validation process to it. The errors in the two failing samples concerned null pointers during intermediary steps of VarDict's execution. Considering VarDict was developed to process WES, the errors may have been caused by the considerably larger amount of data processing WGS encompasses. The third aspect was filtering out germline variant calls from the output VCF. This had to be done because VarDict always calls germline variants, even if it is tuned for somatic variant calling.

After successfully running the Somatic Variant Calling module, we analysed each variant caller's results. Three points arose: 1) 17% of the variants from the ground truth were not called because they contained dated annotations – "INV", "DUP", "DEL", "IGN" and "MSK". These annotations were located in the "ALT" column of the ground truth VCF file (refer back to the VCF example shown in Listing 2.2). None of the variant callers annotate variants in such a way. 2) We encountered *multiallelic variants* – variants with more than one observed variation – each represented in a single row of the VCF file output by Mutect2. 3) The ground truth only considered chromosomes 1 to 22 and X.

To address aspect "1)" in the previous paragraph, we decided to filter out all variants other than SNVs from the VCF files. We chose to keep SNVs because these are consistently represented across all VCF files (those output by the variant callers and those representing the ground truth). Concerning aspect "2)", since not all variant callers represent multiallelic variants in a single row, using BCFtools we separated them into multiple rows (now each representing an SNV). Regarding aspect "3)" in the previous paragraph, we discarded every contig from our analysis that was not in chromosomes 1 to 22 and X.

Finally, both the ground truth VCF and the hg19 reference file required sorting to be used with the GATK tool to combine variants from different files. The files were required to follow a natural sorting (instead of alphabetical where, for example, 11 comes before 2). Hence, the ground truth VCF was sorted using Picard's "SortVcf" and the hg19 reference with SAMtools' "faidx".

#### 4.2.1.B Results

With the caveats from Section 4.2.1.A in mind, we proceeded to take statistical measures of the variant calling performance of our pipeline running on the aforementioned synthetic data. We based ourselves on the measures typically used to compare variant callers that are mentioned in Section 2.3. Recogniz-

ing that we are before a binary classification test, a variant called by our pipeline is considered either true or false, depending on whether it is in the ground truth file. Thus, each variant called can be classified as a True Positive (TP), False Positive (FP), False Negative (FN), or True Negative (TN). Table 4.1 shows a confusion matrix, a common way to help understand TP, FP, FN, and TN.

**Table 4.1:** Representation of a confusion matrix.

| Confusion Matrix | | Ground Truth | |
|---|---|---|---|
| | | TRUE | FALSE |
| Predicted | TRUE | TP | FP |
| | FALSE | FN | TN |

In our case, a TP is a variant call that is also found in the ground truth file; a FP is a variant call that was not in the ground truth file; a FN is a variant that is in the ground truth file but that was either not called at all (missed) or that was called but was filtered by the variant caller; a TN is a variant call that was filtered by the caller and that is not in the ground truth file. Consequently, the number of TPs plus FNs totals the number of variants in the ground truth file, which is 3537. Using these four metrics, we can calculate sensitivity (true positive rate) and specificity (true negative rate):

$$Sensitivity = \frac{TP}{Positives} = \frac{TP}{TP + FN} \tag{4.1}$$

$$Specificity = \frac{TN}{Negatives} = \frac{TN}{TN + FP} \tag{4.2}$$

Figure 4.1 shows the number of TPs, FPs, TNs and FNs, the latter subdivided into "(not called+filtered)". Figures 4.1A-C analyse each variant caller individually. In these, the counts were obtained by first merging (with GATK CombineVariants) the VCF file output by the variant caller with the ground truth VCF file, followed by running shell commands that process the resulting file, calculating the number of variants that fit into each category. Figure 4.1D analyses the union of all variant callers' output, following the same logic as before. Figure 4.1E presents the same union analysis but only for Mutect2 and Strelka2. We cover the latter case to assess VarDict's impact as it shows the best sensitivity but also the worst specificity. Appendix C.1 presents the exact values obtained during the validation process. Table 4.2 shows the sensitivity and specificity of each variant caller.

**Table 4.2:** Sensitivity and specificity values of Mutect2, Strelka2 and VarDict running on synthetic data.

| | **Mutect2 (M2)** | **Strelka2 (S2)** | **VarDict (VD)** | **Union M2, S2, VD** | **Union M2, S2** |
|---|---|---|---|---|---|
| **Sensitivity** | 0.973 | 0.957 | 0.987 | 0.991 | 0.980 |
| **Specificity** | 0.981 | 0.995 | 0.791 | 0.839 | 0.992 |

Concerning the values for sensitivity in Table 4.2, we consider the variant callers to have performed

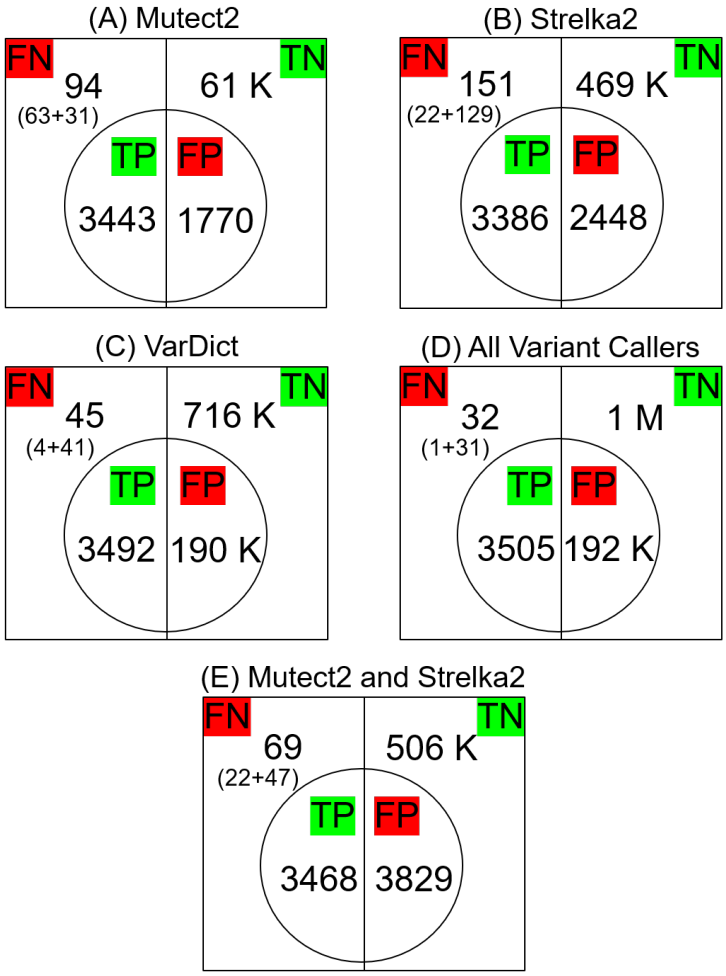


**Figure 4.1:** TP, FP, FN and TN counts from running the newer pipeline's Somatic Variant Calling module on synthetically generated alignment files. These counts were produced in relation to a ground truth file. (A), (B) and (C) concern Mutect2, Strelka2 and VarDict, respectively. (D) presents the counts obtained after doing the union of the called variants from the individual variant callers. Note that FNs are subdivided into "(not called+filtered)" variants. For example, (D) shows that one ground truth variant was not called by any of the variant callers and that 31 ground truth variants, although called, ended up being filtered by some or all the variant callers. Given the high FP count introduced by VarDict, we present (E). It follows the same logic as (D) but only for Mutect2 and Strelka2. In all figures, high values were rounded – followed by "K" or "M", for thousands and millions, respectively.

well in finding the true variants. However, even though VarDict had the best sensitivity, it called around 190 thousand FP variants (see Figure 4.1C), resulting in a subpar value for specificity. For the Pathology group at LUMC, this is not problematic because the priority is finding all possible protein changing variants in CRC with low mutation numbers.

Looking at the FNs in Figure 4.1D, there were 31 ground truth variants being called but filtered by the variants callers and 1 ground truth variant that was not called. Regarding the 31 ground truth variants that are filtered, the variant callers do so either because of low mapping quality or due to the lack of supporting reads.

#### 4.2.1.C Remarks

All in all, the synthetic dataset showed unpractical to use for validation due to the considerable amount of adjustments required, which resulted in only validating the SNV performance of the variant callers. Additionally, we found that some of Mutect2's FPs were variants present in the ground truth file but that were reported as MNVs so, similarly to multiallelic variants, they were shown in a single row of the output VCF file – in total Mutect2 reported 142 MNVs. Moreover, after inspection we found that many of the FP variants were good predictions that should have been included in the ground truth file. We considered this to be another relevant flaw in this dataset and discuss it further, along with the FN definition used, in Section 5.1. Initially, we converted the dataset files to follow the hg38 reference because of its advantages (see Section 2.2.3). After comparing the results with those of the original dataset (using hg19), we concluded that using hg38 would further hinder the validation process (detailed in Appendix C.2).

### 4.2.2 Validation with Generated Synthetic Data

After the validation with the ICGC-TCGA DREAM synthetic data, we tried generating our own synthetic data. For this we chose Xome-Blender [173], a cancer genome simulator that takes real samples as input. We decided to use it because of its claim of not adding any synthetic element. Essentially, this meant that Xome-Blender took the input of a normal individual and generated tumor mutations from it, in a sort of backtracking clonal evolution process. After several attempts in tuning the parameters and restricting the region of processing to the exome, we could not get the tool to run reliably for the normal patient samples we had available. We could not understand the cause of the problem.

### 4.2.3 Validation with Previous Results

A major concern in developing the new pipeline was maintaining coherency with the results from the previous pipeline. This would allow continuity in the analysis of the same samples and give confidence

in the predicting performance of the new pipeline in newer samples. In practice, this meant the new pipeline was required to call the same protein changing variants as the previous one did.

#### 4.2.3.A Methodology

To confirm whether the requisite above was met, we ran the new pipeline on real, low-mutation CRC patient's tumor-normal paired WES samples that had already been processed by the previous version of the pipeline. Moreover, the predicted variants from the previous pipeline had already undergone visual inspection by the Pathology group. Briefly, this inspection verifies whether each variant is non-synonymous (codes for different proteins than originally) and that it is being expressed by checking the corresponding RNA sequencing reads. These variants represent the closest possible to a ground truth, and we refer to them as target variants.

#### 4.2.3.B Results

Table 4.3 summarizes the validation results obtained. The variants called by all variant callers (intersection) are considered true and can skip individual inspection. This is identical to the workflow followed by the Pathology group when using the previous pipeline. In the table we see that, across all samples, 91% of the target variants were in the intersection results. The remaining 9% are divided in two categories: requiring inspection and filtered by all. The former are variants that, in a real patient analysis done by the Pathology group, would have to be individually inspected to assess whether they would be considered true. In these low mutation samples, all the variants are considered even if they are only called by one variant caller. The variants that are not called by the three variant callers (intersection), are visually inspected to discard sequencing artifacts. Furthermore, two of the variants that required inspection (the one in the NIC3 sample and one from NIC7) were correctly reported as MNVs, whereas in the previous pipeline they had been detected as SNVs. Three variants were identified but filtered unanimously by the variant callers (NIC7 sample). The filters were related to either weak evidence, low quality or read bias, or a combination of them. These filters, combined with visual inspection in IGV, lead to discussion of the previous categorization of the variants as true.

#### 4.2.3.C Remarks

We consider the results are in line with the list of variants generated by the previous pipeline. Additionally, if we were to adjust our results to consider the two MNVs as belonging to the intersection, and the three "filtered by all" variants to be excluded from the target variants, the resulting percentage for the intersection would go up to 94.1%. Ideally, more samples should be analysed.

**Table 4.3:** New pipeline's validation results relative to the expressed protein changing variants identified by the previous pipeline. The sample column contains the samples' names and simply serves as an identifier. The target variants column shows the number of expressed protein changing variants previously identified. The next columns contain the number of target variants called by each variant caller in the new pipeline. The intersection presents the number of target variants that were called by all variant callers. The values in the column of (variants) requiring inspection were obtained by subtracting the number of target variants with that of the intersection added to the value in the last column. The last column shows the number of target variants that, although called, were filtered by all variant callers. Values marked with an asterisk (*) include one correctly reported MNV (in the respective sample) that corresponds to an SNV in the previous pipeline. The two asterisks (**) concern three variants that, after a second visual inspection were considered false positives and removed from the target variants list.

| Sample | Target Variants | Mutect2 | Strelka2 | VarDict | Intersection | Require Inspection | Filtered by All |
|---|---|---|---|---|---|---|---|
| NIC3 | 19 | 19 | 19 | 18 | 18 | 1* | 0 |
| NIC4 | 30 | 29 | 30 | 30 | 29 | 1 | 0 |
| NIC5 | 50 | 46 | 49 | 50 | 45 | 5 | 0 |
| NIC6 | 23 | 22 | 23 | 23 | 22 | 1 | 0 |
| NIC7 | 33 | 27 | 29 | 29 | 27 | 3* | 3** |
| **Total** | 155 (100%) | 143 (94%) | 150 (96.8%) | 150 (96.8%) | 141 (91%) | 11 (7.1%) | 3 (1.9%) |

# 5

# Conclusions and Future Work

## Contents

## 5.1 Conclusions

In this work we present a type of bioinformatics pipeline whose purpose is helping treat patients in the field of cancer immunotherapy: the neoantigen identification pipeline. This type of pipeline comprises several computational steps, has varying levels of complexity and may focus only on a subset of all the required steps to identify neoantigens. Specifically, the pipeline we present in this thesis covers all the steps of the typical neoantigen discovery workflow (summarized in Figure 2.1). Succinctly, before executing the pipeline, normal and tumor tissue samples are collected from the patient and sequenced, resulting in millions of nucleotide sequences called reads. Reads are the first main pipeline input, and they initially go through a quality control step. Then, they are pieced together relatively to a reference genome. Now having a map of the reads' locations in the human genome, it is possible to run the crucial steps of identifying tumor-specific variants and the subsequently produced mutant proteins. The last step is predicting which of these proteins are most likely to be recognized by the patient's immune system to aid in fighting the cancer. Conceptually, we divided each step of the pipeline into a module (shown in Figure 3.1).

### 5.1.1 Main Goals

The pipeline presented in this dissertation was developed in the context of the Immunogenomics group from LUMC's Pathology Department. Besides maximizing the number of protein changing variants detected, the pipeline had the two main goals of reducing execution time and maintaining the same predictive performance, when compared to the previously used makefile-based pipeline (see Section 2.9). Both of these goals were accomplished. Concerning reducing execution time, it was accomplished by leveraging the HPC infrastructure available. We saw the execution time being reduced from days to hours. There are no results to support the latter for three reasons: 1) the previous pipeline does not run in the Shark cluster as Makefile jobs are not supported. 2) The previous pipeline was executed in a server used for regular diagnostics. As such, the server could only be used to run the previous pipeline when it was available, i.e., the pipeline was divided into parts that were executed when the server was available. 3) Benchmarking the new pipeline running on an HPC cluster is not trivial. We would have to account for factors such as: queue times for each job; heterogeneous hardware between the nodes where each job is run; the load on the nodes caused by other users; any other bottlenecks (for example, storage and networking devices I/O). Hence, a study outside the scope of our work would have to be done (we expand on this topic in Section 5.2.3). However, to give a rough idea of the integrated pipeline's execution time, it took each real patient WES sample used in the pipeline validation results in Section 4.2.3, between 10 to 14 hours to be processed in the Shark Cluster. Concerning the other main goal of maintaining the same predictive performance as before, we verified whether the new

pipeline called the same protein changing variants as the previous pipeline. Table 4.3 shows that all target variants are called.

### 5.1.2 Further Validation

Furthermore, we validated the variants output by the pipeline on publicly available synthetically generated data (see Section 4.2.1.B). To use it in our validation, several adjustments were required, making its use rather unpractical and limiting the conclusions.

Nonetheless, we consider that the variants called by the union of all variant callers was good, achieving a sensitivity of 0.991 (refer to Table 4.2 to see this and the other results mentioned in this paragraph). However, in the same union we get a specificity of 0.839, negatively skewed by VarDict's high false positive count. In a broader sense, this is a consequence of VarDict calling a disproportionate amount of variants compared to Strelka2 and Mutect2 (see Figures 4.1A-C), the majority of them being TNs. If we remove VarDict from the union, we observe an increase in specificity to 0.992 (approximately 18% increase), while only decreasing the sensitivity to 0.980 (approximately 1% decrease). Although VarDict has the stronger influence in the specificity, looking at Figures 4.1A-C, we notice that both Mutect2 and Strelka2 also have high FP and TN counts. Even though the synthetic dataset in question allowed to validate the pipeline's variant calling performance, we conclude (and address below) that some aspects of this dataset were not ideal.

Let us recall that we defined TNs as variant calls that were filtered by the respective variant caller. This was because the ground truth VCF file only included the variants that were injected into the synthetic data as true variants. Hence, there were no variants labeled as false in the ground truth. Having no variants labeled as false has at least two implications: 1) specificity can be meaninglessly increased by an high TN count, provided the value of FPs is of a smaller magnitude. 2) More importantly, it is possible that real true variants are called that are not in the ground truth file, consequently affecting the assessment of the tools' variant calling performance. We saw that this happened for many of the detected FP variants. However, we considered there was little benefit in going through the thousands of FPs because not only were we more interested in maximising the sensitivity (detect as much true variants as possible) but also because even if we concluded that the majority of the FPs were good predictions, we still could not add the latter to the ground truth file as we needed an objective ground truth. Since the FP count could not be trusted, we did not present other metrics of performance, such as the Positive Predictive Value (PPV). PPV measures the proportion of positive results, given by $TP/(TP+FP)$. The latter would show us the proportion of variants that are called correctly as true.

Additionally, we tried generating synthetic data ourselves where we would have more control over the injected variants and original samples. However, the time invested in this was fruitless as the tool chosen was not capable of processing our patient samples (see Section 4.2.2). We decided to discard

this idea because the comparison with the previous pipeline version results was considered enough.

### 5.1.3 Integrated Pipeline

The integrated pipeline [17] successfully runs in the Shark HPC Cluster (see Section 3.2) and should be able to run in other HPC environments provided the backend is properly configured (as exemplified for SGE in Appendix A.1). To run the pipeline, a simple CLI command taking as parameter an input file is run (see Section 3.7.4). The inputs passed on to the pipeline before execution reside in a single JSON file that can be automatically generated using the Woomtool (see Section 3.7.3). This allows easy input definition but is also verbose, which can be overwhelming to a new user (see Listing 3.10). This verboseness is inherent to having multiple WDL (sub)workflows being called, which leads to long input variables. WDL, being a language designed to build workflows, natively provides useful constructs such as a parallelization mechanism (scatter), helping reduce execution time, and "File" type variables that simplify handling files paths in the code. In general, and specifically comparing to the monolithic makefile approach of the the previous pipeline, WDL scripts favor readability, making them more maintainable and motivating contribution from the bioinformatics community. WDL's natural division into workflows favors modularity. To the same end, the integrated pipeline is divided into git submodules that can be easily integrated in other git projects or used standalone (see Section 3.6).

WDL workflows are interpreted by the Cromwell WFMS (see Section 3.4). Cromwell manages job scheduling and supports containerization software, allowing result reproducibility. Singularity is the tool used to pull and build container images because it is what the Shark Cluster supports for this purpose (see Section 3.5). Singularity is compatible with the more widely used Docker images, so we found that every tool needed in the pipeline was already containerized. These tools were chosen based on the research presented in Chapter 2. The most notable tools per module are: FastQC, MultiQC and Cutadapt for QC and Adapter Trimming; BWA-MEM for Read Mapping; Mutect2, Strelka2 and VarDict for Somatic Variant Calling. Picard, SAMtools and the GATK's toolkit are noteworthy as these tools were used in diverse intermediary steps, saving us development time by addressing common data processing tasks. We summarize each modules' workflows using UML sequence diagrams (see Figures 3.4, 3.6 and 3.7). We highlight our decision to include the Read Orientation Artifacts and Panel of Normals procedures to Mutect2's workflow, both contributing to variant filtering. The former calculates read related biases and the latter takes into account variants found in the normal samples of other patients. Lastly, continuous integration is assured using Travis CI and the pytest framework, and all software used and developed in this thesis is open source and can be used without a fee.

### 5.1.4 Standalone Pipeline Modules

The three remaining modules – Mutant Protein Prediction, HLA Typing and HLA Binding Prediction – are run separately from the integrated pipeline. The tools used in these modules were supported by the research presented in Chapter 2, of which some were also already used in the previous pipeline. When compared to the previous pipeline, the first two modules were not modified. Regarding the Mutant Protein Prediction module (see Section 3.9.1), it was initially planned to be extended by us but it started being actively developed in the same period of our work. Regarding the HLA Typing module (see Section 3.9.2), there was never a plan to modify it due to a previous positive experience with it. Consequently, we chose to focus our efforts in the remaining module, HLA Binding Prediction, for which we developed a Python tool [152], referred to as bind-pred in this thesis (see Section 3.9.3).

For bind-pred, we first modified the interface tool that runs the HLA binding prediction tools [155]. With it we were able to obtain an additional type of peptide ranking besides the provided in the original tool. For bind-pred to function as intended, it was essential that the alignments between MUT and WT peptides were done correctly. We ended up with the gap function shown in Listing 3.15, that provided correct alignments for all the different cases we encountered. Furthermore, we used a novel approach that filters MUT peptides found to be exact matches when querying a peptide database. The intention is to exclude normal peptides, i.e., peptides that are usually being produced in healthy tissue. This way we filter out peptides that should not have potential in helping treating tumors. Besides peptide rankings, bind-pred outputs two other files auxiliary to lab work (mass spectrometry and peptide reactivity testing).

### 5.1.5 User Interface

An R Shiny application was extended to plot the number of protein changing variants per sample and the type of variants (see Section 3.10). It is available in GitHub [163] and online [164]. However, it needs further development to be used outside the context of the Pathology department at LUMC. Nonetheless, it helped identify a problem with the sample data (see Section 4.1.4), demonstrating the power of data visualization.

## 5.2 Future Work

In this section we address topics that could be explored to extend and improve the presented work.

### 5.2.1 Results Database

As the number of patients analysed by the pipeline increases, a Database (DB), possibly tied to the HPC Cluster storage, would provide easier querying of the generated results, facilitating subsequent

analyses. A DB would also result in a more structured way of storing the results, rather than relying on shared directories in the installed network file system (current procedure). Lastly, a DB could provide data consistency and durability, avoiding storing invalid data and corrupting data in the case of a system failure. Research about the supported and recommended DB system for the general case and for running on the Shark cluster would have to be done but given the specific formats of data generated by this type of pipelines, specialized DB frameworks such as Gemini [174] could be considered. Besides not having to put as much development effort into designing a DB solution, specific advantages of using Gemini are that it allows to directly load VCF files into the DB and extends the data with annotations from trustworthy sources such as the ones from the ENCODE project [175]. Additionally, and particularly for LUMC, a collaboration with the Department of Clinical Genetics which already has a DB solution in place to store germline variants could be done.

### 5.2.2 User Interface

Although a CLI is used to run the pipeline, having the ability to interact with the latter through a Graphical User Interface (GUI) makes the developed software more enticing for users that may not be CLI savvy (such as biologists without a background in information technologies). A GUI can also offer a space for the user to observe and interact with the results collected from the analysis, e.g., explore the positions of the genome where phased tumor variants occur. The latter can be done by generating a hyperlink to an open IGV session (with port listening enabled), something that is straightforward and documented [176]. Furthermore, when compared to a CLI, a GUI can provide a more familiar and user-friendly interface to set the pipeline's input parameters and simplify accessing, locating and moving the resulting data. With the GUI there is the potential to open the pipeline to a considerably wider range of researchers, as in practice many are still reluctant to operate a CLI if they have little to no previous experience with it.

An appropriate tool to build a simple GUI is Tkinter [177]: the standard Python interface to Tk [178], a toolkit that provides a library of basic elements of GUI widgets. Tkinter comes included with most binary distributions of Python and is compatible with Unix and Windows platforms. These properties make it a good choice for a cross-platform implementation, which is important given the widespread use of the Windows OS in research laboratories (as is the case at LUMC). Although the pipeline only runs in Unix platforms, this GUI could be run in Windows systems because the main requirement is to be able to connect to the HPC cluster using Secure Shell (SSH) which, especially in the current Windows OS version (Windows 10), is a simple task to accomplish. For example, by installing the Windows Subsystem for Linux, currently already possible through the integrated application store. Another alternative, and taking into account that the Shark cluster comes with Python installed, the GUI could instead be installed on the cluster and then have the application display forwarded to the Windows machine by using X11 SSH forwarding. However, this solution would require more configuration. For local installations of the

pipeline, e.g., in a researcher's laptop, the Tkinter GUI would necessarily have to be installed in a Unix platform as it is the one supported by the pipeline. Alternatively to Tkinter, the R Shiny module could be further extended, which has as the advantage of being easily deployed on-premises or in the cloud for free [179].

### 5.2.3 Benchmarking Performance

Although having verified a significant reduction in execution time when compared to the previously used pipeline, further optimization could be accomplished. Benchmarking and code profiling can provide insights into current sources of bottlenecks in the pipeline.

As alluded to in Section 5.1.1, benchmarking the pipeline running in any HPC cluster poses some problems. Typically users do not have exclusive usage of the computing nodes, thus disk Input/Output (I/O) is usually the main limiting factor in performance. There is also added complexity because the computing nodes are not homogeneous regarding the hardware installed. This affects, among other aspects, the ability to make meaningful comparisons between different pipeline runs. Nonetheless, measures related to Central Processing Unit (CPU) and primary memory usage, and the amount of data transferred in I/O operations can be collected by querying the Shark cluster with the *qacct* command [180]. Moreover, a profiling tool can then help to identify the sections of code to optimize. Also, the pipeline's modularity can in part mitigate the problems mentioned above, by allowing for shorter, more manageable runs. Furthermore, comparing performance with other pipelines is also a point of interest to assess potential improvements that can be done.

### 5.2.4 Integrating Standalone Modules

Succinctly, integrating the remaining pipelines would involve containerizing Isovar and the developed tools for the HLA Binding Prediction module. Moreover, the process between the Somatic Variant Calling and the Mutant Protein Prediction modules would have to be streamlined, as there is at least a visual inspection step that has to be done (that could possibly be avoided by modifying Isovar directly). Finally, the corresponding WDL workflows and integration tests would have to be written.

### 5.2.5 Tracking Code Coverage

To track code coverage - percentage of source-code covered by tests - over time, the web service Coveralls [181] could be used. It officially supports Travis CI (see Section 3.11), is programming language independent, and free for open source projects.

# Bibliography


[1] "Akka-http Homepage," https://doc.akka.io/docs/akka-http/current/index.html, last accessed 19 Oct 2020.

[2] S. Behjati and P. S. Tarpey, "What is next generation sequencing?" *Archives of Disease in Childhood - Education and Practice*, vol. 98, no. 6, pp. 236–238, 2013. [Online]. Available: https://doi.org/10.1136/archdischild-2013-304340

[3] L. C. Brody, "Nucleotide," https://www.genome.gov/genetics-glossary/Nucleotide, last accessed 21 Oct 2019.

[4] L. Liu, Y. Li, S. Li, N. Hu, Y. He, R. Pong, D. Lin, L. Lu, and M. Law, "Comparison of next-generation sequencing systems," *Journal of biomedicine & biotechnology*, vol. 2012, pp. 251 364–251 364, 2012. [Online]. Available: https://doi.org/10.1155/2012/251364

[5] M. A. Quail, M. Smith, P. Coupland, T. D. Otto, S. R. Harris, T. R. Connor, A. Bertoni, H. P. Swerdlow, and Y. Gu, "A tale of three next generation sequencing platforms: comparison of ion torrent, pacific biosciences and illumina miseq sequencers," *BMC genomics*, vol. 13, pp. 341–341, Jul 2012. [Online]. Available: https://doi.org/10.1186/1471-2164-13-341

[6] Illumina, "Key Illumina Technology Overviews," https://www.illumina.com/science/technology.html, last accessed 17 Dec 2018.

[7] "Genome Research Consortium," https://www.ncbi.nlm.nih.gov/grc, last accessed 30 Sep 2019.

[8] E. F. Fritsch, M. Rajasagi, P. A. Ott, V. Brusic, N. Hacohen, and C. J. Wu, "HLA-binding properties of tumor neoepitopes in humans," *Cancer Immunology Research*, vol. 2, no. 6, pp. 522–529, 2014. [Online]. Available: https://doi.org/10.1158/2326-6066.CIR-13-0227

[9] J. Finnigan, A. Rubinsteyn, J. Hammerbacher, and N. Bhardwaj, "Mutation-derived tumor antigens: Novel targets in cancer immunotherapy," *Oncology (Williston Park, N.Y.)*, vol. 29, 2015.

[10] A. Wilm, P. P. K. Aw, D. Bertrand, G. H. T. Yeo, S. H. Ong, C. H. Wong, C. C. Khor, R. Petric, M. L. Hibberd, and N. Nagarajan, "Lofreq: a sequence-quality aware, ultra-sensitive

variant caller for uncovering cell-population heterogeneity from high-throughput sequencing datasets," *Nucleic Acids Res*, vol. 40, no. 22, pp. 11 189–11 201, 2012. [Online]. Available: https://doi.org/10.1093/nar/gks918

[11] S. Maleki Vareki, "High and low mutational burden tumors versus immunologically hot and cold tumors and response to immune checkpoint inhibitors," *Journal for immunotherapy of cancer*, vol. 6, no. 1, pp. 157–157, Dec 2018. [Online]. Available: https://doi.org/10.1186/s40425-018-0479-7

[12] P. Saxová, S. Buus, S. Brunak, and C. Keşmir, "Predicting proteasomal cleavage sites: a comparison of available methods," *International Immunology*, vol. 15, no. 7, pp. 781–787, 2003. [Online]. Available: https://doi.org/10.1093/intimm/dxg084

[13] P. A. Ott, Z. Hu, D. B. Keskin, S. A. Shukla, J. Sun, D. J. Bozym, W. Zhang, A. Luoma, A. Giobbie-Hurder, L. Peter, C. Chen, O. Olive, T. A. Carter, S. Li, D. J. Lieb, T. Eisenhaure, E. Gjini, J. Stevens, W. J. Lane, I. Javeri, K. Nellaiappan, A. M. Salazar, H. Daley, M. Seaman, E. I. Buchbinder, C. H. Yoon, M. Harden, N. Lennon, S. Gabriel, S. J. Rodig, D. H. Barouch, J. C. Aster, G. Getz, K. Wucherpfennig, D. Neuberg, J. Ritz, E. S. Lander, E. F. Fritsch, N. Hacohen, and C. J. Wu, "An immunogenic personal neoantigen vaccine for patients with melanoma," *Nature*, vol. 547, pp. 217–221, 2017. [Online]. Available: https://doi.org/10.1038/nature22991

[14] U. Sahin, E. Derhovanessian, M. Miller, B.-P. Kloke, P. Simon, M. Löwer, V. Bukur, A. D. Tadmor, U. Luxemburger, B. Schrörs, T. Omokoko, M. Vormehr, C. Albrecht, A. Paruzynski, A. N. Kuhn, J. Buck, S. Heesch, K. H. Schreeb, F. Müller, I. Ortseifer, I. Vogler, E. Godehardt, S. Attig, R. Rae, A. Breitkreuz, C. Tolliver, M. Suchan, G. Martic, A. Hohberger, P. Sorn, J. Diekmann, J. Ciesla, O. Waksmann, A.-K. Brück, M. Witt, M. Zillgen, A. Rothermel, B. Kasemann, D. Langer, S. Bolte, M. Diken, S. Kreiter, R. Nemecek, C. Gebhardt, S. Grabbe, C. Höller, J. Utikal, C. Huber, C. Loquai, and Ö. Türeci, "Personalized RNA mutanome vaccines mobilize poly-specific therapeutic immunity against cancer," *Nature*, vol. 547, pp. 222–226, 2017. [Online]. Available: https://doi.org/10.1038/nature23003

[15] Docker Inc., "Enterprise Container Platform," https://www.docker.com/, last accessed 21 Oct 2019.

[16] "Immunogenomics Group, Pathology Department, LUMC," https://www.lumc.nl/org/pathologie/research/90708043159185/1709335/, last accessed 23 Nov 2020.

[17] R. Vorderman, P. van 't Hof, D. Cats, A. Paulo, and L. Mei, "biowdl/germline-dna: Release 1.0.0," https://doi.org/10.5281/zenodo.3459263, Sep 2019.

[18] R. Vorderman, D. Cats, P. van 't Hof, C. Agaser, A. Paulo, and L. Mei, "biowdl/germline-dna: Release 4.0.0," https://doi.org/10.5281/zenodo.3974515, Aug 2020.

[19] K. Voss, J. Gentry, and G. Van der Auwera, “Full-stack genomics pipelining with GATK4 + WDL + Cromwell [version 1; not peer reviewed],” *F1000Research*, vol. 6(ISCB Comm J):1379 (poster), 2017. [Online]. Available: https://doi.org/10.7490/f1000research.1114631.1

[20] R Studio, “R Shiny,” https://shiny.rstudio.com/, last accessed 21 Oct 2019.

[21] V. I. Jurtz and L. R. Olsen, *Cancer Bioinformatics*, ser. Methods in Molecular Biology. New York, NY: Humana Press, 2019, vol. 1878, ch. 9, pp. 157–172. [Online]. Available: https://doi.org/10.1007/978-1-4939-8868-6_9

[22] Broad Institute, “Introduction to the GATK Best Practices,” https://software.broadinstitute.org/gatk/best-practices/, last accessed 21 Dec 2018.

[23] Babraham Bioinformatics, “FastQC Homepage,” https://www.bioinformatics.babraham.ac.uk/projects/fastqc/, last accessed 19 Dec 2018.

[24] B. Ewing and P. Green, “Base-calling of automated sequencer traces using phred. II. Error probabilities,” *Genome Res.*, vol. 8, no. 3, pp. 186–194, Mar 1998.

[25] C. Del Fabbro, S. Scalabrin, M. Morgante, and F. M. Giorgi, “An extensive evaluation of read trimming effects on illumina ngs data analysis,” *PLOS ONE*, vol. 8, no. 12, 2013. [Online]. Available: https://doi.org/10.1371/journal.pone.0085024

[26] M. Martin, “Cutadapt removes adapter sequences from high-throughput sequencing reads,” *EMBnet.journal*, vol. 17, no. 1, pp. 10–12, 2011. [Online]. Available: http://journal.embnet.org/index.php/embnetjournal/article/view/200

[27] R. Knut, B. Langmead, D. Weese, and D. Evers, “Alignment of next-generation sequencing reads,” *Annual review of genomics and human genetics*, vol. 16, 2015. [Online]. Available: https://doi.org/10.1146/annurev-genom-090413-025358

[28] U. Manber and G. Myers, “Suffix arrays: A new method for on-line string searches,” *SIAM Journal on Computing*, vol. 22, no. 5, p. 935–948, 1993. [Online]. Available: https://doi.org/10.1137/0222058

[29] M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch, “Replacing suffix trees with enhanced suffix arrays,” *Journal of Discrete Algorithms*, vol. 2, no. 1, pp. 53–86, 2004.

[30] P. Ferragina and G. Manzini, “Opportunistic data structures with applications,” in *Proceedings 41st Annual Symposium on Foundations of Computer Science*, 2000, pp. 390–398. [Online]. Available: https://doi.org/10.1109/SFCS.2000.892127

[31] M. Burrows and D. J. Wheeler, “A block-sorting lossless data compression algorithm,” *Digital Systems Research Center Research Reports*, vol. 1, 1995.

[32] H. Li and R. Durbin, “Fast and accurate short read alignment with burrows–wheeler transform,” *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009. [Online]. Available: https://doi.org/10.1093/bioinformatics/btp324

[33] B. Langmead and S. L. Salzberg, “Fast gapped-read alignment with bowtie 2,” *Nature Methods*, vol. 9, pp. 357–359, 2012. [Online]. Available: https://doi.org/10.1038/nmeth.1923

[34] A. Dobin, C. A. Davis, F. Schlesinger, J. Drenkow, C. Zaleski, S. Jha, P. Batut, M. Chaisson, and T. R. Gingeras, “STAR: ultrafast universal RNA-seq aligner,” *Bioinformatics*, vol. 29, no. 1, pp. 15–21, 10 2012. [Online]. Available: https://doi.org/10.1093/bioinformatics/bts635

[35] D. Kim, G. Pertea, C. Trapnell, H. Pimentel, R. Kelley, and S. L. Salzberg, “Tophat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions,” *Genome Biology*, vol. 14, no. 4, p. R36, 2013. [Online]. Available: https://doi.org/10.1186/gb-2013-14-4-r36

[36] N. L. Bray, H. Pimentel, P. Melsted, and L. Pachter, “Near-optimal probabilistic rna-seq quantification,” *Nature Biotechnology*, vol. 34, pp. 525–527, 2016. [Online]. Available: https://doi.org/10.1038/nbt.3519

[37] K. Belk, C. Boucher, A. Bowe, T. Gagie, P. Morley, M. D. Muggli, N. R. Noyes, S. J. Puglisi, and R. Raymond, “Succinct colored de bruijn graphs,” *bioRxiv*, 2016. [Online]. Available: https://www.biorxiv.org/content/early/2016/02/18/040071

[38] European Bioinformatics Institute , “EMBL-EBI Homepage,” https://www.ebi.ac.uk/, last accessed 21 Dec 2018.

[39] European Bioinformatics Institute, “HTS Mappers,” https://bit.ly/2P7lEDQ, last accessed 21 Dec 2018.

[40] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, and 1000 Genome Project Data Processing Subgroup, “The Sequence Alignment/Map format and SAMtools,” *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 06 2009. [Online]. Available: https://doi.org/10.1093/bioinformatics/btp352

[41] D. Barnett, “BAMtools Github Repository,” https://github.com/pezmaster31/bamtools, last accessed 26 Aug 2020.

[42] Broad Institute, “Picard Tool Homepage,” https://broadinstitute.github.io/picard/, last accessed 26 Aug 2020.

[43] C. Xu, “A review of somatic single nucleotide variant calling algorithms for next-generation sequencing data,” *Comput Struct Biotechnol J*, vol. 16, pp. 15–24, 2018. [Online]. Available: https://doi.org/10.1016/j.csbj.2018.01.003

[44] M. R. Stratton, P. J. Campbell, and P. A. Futreal, “The cancer genome,” *Nature*, vol. 458, no. 7239, pp. 719–724, 2009. [Online]. Available: https://doi.org/10.1038/nature07943

[45] K. L. Gorringe, *Loss of Heterozygosity*. American Cancer Society, 2016, pp. 1–8. [Online]. Available: https://doi.org/10.1002/9780470015902.a0026643

[46] C. T Saunders, W. Wong, S. Swamy, J. Becq, L. J Murray, and K. Cheetham, “Strelka: Accurate somatic small-variant calling from sequenced tumor-normal sample pairs,” *Bioinformatics (Oxford, England)*, vol. 28, pp. 1811–7, 2012. [Online]. Available: https://doi.org/10.1093/bioinformatics/bts271

[47] Broad Institute, “Mutect2 Homepage,” https://gatk.broadinstitute.org/hc/en-us/articles/360037593851-Mutect2, last accessed 26 Aug 2020.

[48] L. T. Fang, P. T. Afshar, A. Chhibber, M. Mohiyuddin, Y. Fan, J. C. Mu, G. Gibeling, S. Barr, N. B. Asadi, M. B. Gerstein, D. C. Koboldt, W. Wang, W. H. Wong, and H. Y. Lam, “An ensemble approach to accurately detect somatic mutations using somaticseq,” *Genome Biology*, vol. 16, no. 1, p. 197, 2015. [Online]. Available: https://doi.org/10.1186/s13059-015-0758-2

[49] A. Rimmer, H. Phan, I. Mathieson, Z. Iqbal, S. Twigg, A. O M Wilkie, G. McVean, and G. Lunter, “Integrating mapping-, assembly- and haplotype-based approaches for calling variants in clinical sequencing applications,” *Nature genetics*, vol. 46, pp. 912–918, 2014. [Online]. Available: https://doi.org/10.1038/ng.3036

[50] A. Christoforides, J. Carpten, G. J Weiss, M. Demeure, D. Von Hoff, and D. Craig, “Identification of somatic mutations in cancer through bayesian-based analysis of sequenced genome pairs,” *BMC genomics*, vol. 14, p. 302, 2013. [Online]. Available: https://doi.org/10.1186/1471-2164-14-302

[51] Z. Lai, A. Markovets, M. Ahdesmaki, B. Chapman, O. Hofmann, R. McEwen, J. Johnson, B. Dougherty, C. Barrett, and J. Dry, “Vardict: A novel and versatile variant caller for next-generation sequencing in cancer research,” *Nucleic Acids Research*, vol. 44, p. e108, 2016. [Online]. Available: https://doi.org/10.1093/nar/gkw227

[52] D. E. Larson, C. C. Harris, K. Chen, D. C. Koboldt, T. E. Abbott, D. J. Dooling, T. J. Ley, E. R. Mardis, R. K. Wilson, and L. Ding, “Somaticsniper: identification of somatic point mutations in whole genome sequencing data,” *Bioinformatics*, vol. 28, no. 3, pp. 311–317, 2012. [Online]. Available: https://doi.org/10.1093/bioinformatics/btr665

[53] W. Wang, P. Wang, F. Xu, R. Luo, M. P. Wong, T.-W. Lam, and J. Wang, "Fasd-somatic: a fast and accurate somatic snv detection algorithm for cancer genome sequencing data," *Bioinformatics*, vol. 30, no. 17, pp. 2498–2500, 2014. [Online]. Available: https://doi.org/10.1093/bioinformatics/btu338

[54] S. Kim, K. Jeong, K. Bhutani, J. H. Lee, A. Patel, E. Scott, H. Nam, H. Lee, J. Gleeson, and V. Bafna, "Virmid: Accurate detection of somatic mutations with sample impurity inference," *Genome biology*, vol. 14, p. R90, 2013. [Online]. Available: https://doi.org/10.1186/gb-2013-14-8-r90

[55] A. Roth, J. Ding, R. Morin, A. Crisan, G. Ha, R. Giuliany, A. Bashashati, M. Hirst, G. Turashvili, A. Oloumi, M. Marra, S. Aparicio, and S. P Shah, "Jointsnvmix : A probabilistic model for accurate detection of somatic mutations in normal/tumour paired next generation sequencing data," *Bioinformatics (Oxford, England)*, vol. 28, pp. 907–13, 2012. [Online]. Available: https://doi.org/10.1093/bioinformatics/bts053

[56] Y. Liu, M. Loewer, S. Aluru, and B. Schmidt, "Snvsniffer: An integrated caller for germline and somatic single-nucleotide and indel mutations," *BMC Systems Biology*, vol. 10, p. 47, 2016. [Online]. Available: https://doi.org/10.1186/s12918-016-0300-5

[57] K. Cibulskis, M. S. Lawrence, S. L. Carter, A. Sivachenko, D. Jaffe, C. Sougnez, S. Gabriel, M. Meyerson, E. S. Lander, and G. Getz, "Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples," *Nature Biotechnology*, vol. 31, pp. 213–219, 2013. [Online]. Available: https://doi.org/10.1038/nbt.2514

[58] Y. Shiraishi, Y. Sato, K. Chiba, Y. Okuno, Y. Nagata, K. Yoshida, N. Shiba, Y. Hayashi, H. Kume, Y. Homma, M. Sanada, S. Ogawa, and S. Miyano, "An empirical Bayesian framework for somatic mutation detection from cancer genome sequencing data," *Nucleic Acids Research*, vol. 41, no. 7, p. e89, 2013. [Online]. Available: https://doi.org/10.1093/nar/gkt126

[59] M. Gerstung, C. Beisel, M. Rechsteiner, P. Wild, P. Schraml, H. Moch, and N. Beerenwinkel, "Reliable detection of subclonal single-nucleotide variants in tumor cell populations," *Nature communications*, vol. 3, p. 811, 2012. [Online]. Available: https://doi.org/10.1038/ncomms1814

[60] J. Carrot-Zhang and J. Majewski, "Lolopicker: detecting low allelic-fraction variants from low-quality cancer samples," *Oncotarget*, vol. 8, no. 23, pp. 37 032–37 040, 2017. [Online]. Available: https://doi.org/10.18632/oncotarget.16144

[61] Y. Fan, L. Xi, D. S. T. Hughes, J. Zhang, J. Zhang, P. A. Futreal, D. A. Wheeler, and W. Wang, "Muse: accounting for tumor heterogeneity using a sample-specific error model improves

sensitivity and specificity in mutation calling from sequencing data," *Genome Biology*, vol. 17, no. 1, p. 178, 2016. [Online]. Available: https://doi.org/10.1186/s13059-016-1029-6

[62] H. Xu, J. DiCarlo, R. V. Satya, Q. Peng, and Y. Wang, "Comparison of somatic mutation calling methods in amplicon and whole exome sequence data," *BMC Genomics*, vol. 15, no. 1, pp. 244–253, 2014. [Online]. Available: https://doi.org/10.1186/1471-2164-15-244

[63] C. Xu, M. R. Nezami Ranjbar, Z. Wu, J. DiCarlo, and Y. Wang, "Detecting very low allele fraction variants using targeted dna sequencing and a novel molecular barcode-aware variant caller," *BMC Genomics*, vol. 18, no. 1, p. 5, Jan 2017. [Online]. Available: https://doi.org/10.1186/s12864-016-3425-4

[64] D. C. Koboldt, Q. Zhang, D. E. Larson, D. Shen, M. D. McLellan, L. Lin, C. A. Miller, E. R. Mardis, L. Ding, and R. K. Wilson, "Varscan 2: Somatic mutation and copy number alteration discovery in cancer by exome sequencing," *Genome Research*, vol. 22, pp. 568–576, 2012. [Online]. Available: https://doi.org/10.1101/gr.129684.111

[65] A. Rubinsteyn, J. Kodysh, I. Hodes, S. Mondet, B. A. Aksoy, J. P. Finnigan, N. Bhardwaj, and J. Hammerbacher, "Computational pipeline for the PGV-001 neoantigen vaccine trial," *Frontiers in Immunology*, vol. 8, p. 1807, 2018. [Online]. Available: https://doi.org/10.3389/fimmu.2017.01807

[66] D. H. Spencer, M. Tyagi, F. Vallania, A. J. Bredemeyer, J. D. Pfeifer, R. D. Mitra, and E. J. Duncavage, "Performance of common analysis methods for detecting low-frequency single nucleotide variants in targeted next-generation sequence data," *The Journal of Molecular Diagnostics*, vol. 16, no. 1, pp. 75 – 88, 2014. [Online]. Available: https://doi.org/10.1016/j.jmoldx.2013.09.003

[67] S. Sengupta, K. Gulukota, Y. Zhu, C. Ober, K. Naughton, W. Wentworth-Sheilds, and Y. Ji, "Ultra-fast local-haplotype variant calling using paired-end dna-sequencing data reveals somatic mosaicism in tumor and normal blood samples," *Nucleic Acids Res*, vol. 44, 09 2015. [Online]. Available: https://doi.org/10.1093/nar/gkv953

[68] P. Danecek, A. Auton, G. Abecasis, C. A. Albers, E. Banks, M. A. DePristo, R. E. Handsaker, G. Lunter, G. T. Marth, S. T. Sherry, G. McVean, R. Durbin, and . G. P. A. Group, "The variant call format and VCFtools," *Bioinformatics*, vol. 27, no. 15, pp. 2156–2158, 06 2011. [Online]. Available: https://doi.org/10.1093/bioinformatics/btr330

[69] Samtools, "Bcftools Homepage," https://samtools.github.io/bcftools/, last accessed 23 Oct 2019.

[70] A. Rubinsteyn, J. Kodysh, and B. A. Aksoy, "hammerlab/isovar: Version 0.7.0," 2017.

[71] EMBL-EBI, “Ensembl Homepage,” https://www.ensembl.org/index.html, last accessed 01 Dec 2020.

[72] J. Harrow, A. Frankish, J. M. Gonzalez, E. Tapanari, M. Diekhans, F. Kokocinski, B. L. Aken, D. Barrell, A. Zadissa, S. Searle, I. Barnes, A. Bignell, V. Boychenko, T. Hunt, M. Kay, G. Mukherjee, J. Rajan, G. Despacio-Reyes, G. Saunders, C. Steward, R. Harte, M. Lin, C. Howald, A. Tanzer, T. Derrien, J. Chrast, N. Walters, S. Balasubramanian, B. Pei, M. Tress, J. M. Rodriguez, I. Ezkurdia, J. van Baren, M. Brent, D. Haussler, M. Kellis, A. Valencia, A. Reymond, M. Gerstein, R. Guigó, and T. J. Hubbard, “GENCODE: the reference human genome annotation for The ENCODE Project,” *Genome Res.*, vol. 22, no. 9, pp. 1760–1774, 2012. [Online]. Available: https://www.doi.org/10.1101/gr.135350.111

[73] W. McLaren, L. Gil, S. E. Hunt, H. S. Riat, G. R. S. Ritchie, A. Thormann, P. Flicek, and F. Cunningham, “The ensembl variant effect predictor,” *Genome Biology*, vol. 17, no. 1, p. 122, Jun 2016. [Online]. Available: https://doi.org/10.1186/s13059-016-0974-4

[74] K. Wang, M. Li, and H. Hakonarson, “Annovar: functional annotation of genetic variants from high-throughput sequencing data,” *Nucleic Acids Research*, vol. 38, no. 16, p. e164, 2010. [Online]. Available: https://doi.org/10.1093/nar/gkq603

[75] D. Middleton, F. Gonzalez, M. Fernandez-Vina, J.-M. Tiercy, S. G. E. Marsh, M. Aubrey, M. G. Bicalho, A. Canossi, V. Carter, S. Cate, F. R. Guerini, P. Loiseau, M. Martinetti, M. E. Moraes, V. Morales, J. Perasaari, M. Setterholm, M. Sprague, S. Tavoularis, M. Torres, S. Vidal, C. Witt, G. Wohlwend, and K. L. Yang, “A bioinformatics approach to ascertaining the rarity of hla alleles,” *Tissue Antigens*, vol. 74, no. 6, pp. 480–485, 2009. [Online]. Available: https://doi.org/10.1111/j.1399-0039.2009.01361.x

[76] S. J. Mack, P. Cano, J. A. Hollenbach, J. He, C. K. Hurley, D. Middleton, M. E. Moraes, S. E. Pereira, J. H. Kempenich, E. F. Reed, M. Setterholm, A. G. Smith, M. G. Tilanus, M. Torres, M. D. Varney, C. E. M. Voorter, G. F. Fischer, K. Fleischhauer, D. Goodridge, W. Klitz, A.-M. Little, M. Maiers, S. G. E. Marsh, C. R. Müller, H. Noreen, E. H. Rozemuller, A. Sanchez-Mazas, D. Senitzer, E. Trachtenberg, and M. Fernandez-Vina, “Common and well-documented hla alleles: 2012 update to the cwd catalogue,” *Tissue Antigens*, vol. 81, no. 4, pp. 194–203, 2013. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/tan.12093

[77] A. Szolek, B. Schubert, C. Mohr, M. Sturm, M. Feldhahn, and O. Kohlbacher, “Optitype: precision hla typing from next-generation sequencing data,” *Bioinformatics*, vol. 30, no. 23, pp. 3310–3316, 2014. [Online]. Available: https://doi.org/10.1093/bioinformatics/btu548

[78] D. C. Bauer, A. Zadoorian, L. O. W. Wilson, M. G. H. Alliance, and N. P. Thorne, "Evaluation of computational programs to predict hla genotypes from genomic sequencing data," *Briefings in Bioinformatics*, vol. 19, no. 2, pp. 179–187, 2018. [Online]. Available: https://doi.org/10.1093/bib/bbw097

[79] N. Nariai, K. Kojima, S. Saito, T. Mimori, Y. Sato, Y. Kawai, Y. Yamaguchi-Kabata, J. Yasuda, and M. Nagasaki, "Hla-vbseq: accurate hla typing at full resolution from whole-genome sequencing data," *BMC Genomics*, vol. 16, no. 2, p. S7, 2015. [Online]. Available: https://doi.org/10.1186/1471-2164-16-S2-S7

[80] Y. Bai, M. Ni, B. Cooper, Y. Wei, and W. Fury, "Inference of high resolution hla types using genome-wide rna or dna sequencing reads," *BMC Genomics*, vol. 15, no. 1, p. 325, 2014. [Online]. Available: https://doi.org/10.1186/1471-2164-15-325

[81] S. Boegel, M. Löwer, M. Schäfer, T. Bukur, J. de Graaf, V. Boisguérin, Ö. Türeci, M. Diken, J. C. Castle, and U. Sahin, "HLA typing from RNA-Seq sequence reads," *Genome Medicine*, vol. 4, no. 12, p. 102, 2012. [Online]. Available: https://doi.org/10.1186/gm403

[82] M. Andreatta and M. Nielsen, "Gapped sequence alignment using artificial neural networks: application to the MHC class I system," *Bioinformatics*, vol. 32, no. 4, pp. 511–517, 2016. [Online]. Available: https://doi.org/10.1093/bioinformatics/btv639

[83] M. Nielsen and M. Andreatta, "NetMHCpan-3.0; improved prediction of binding to MHC class I molecules integrating information from multiple receptor and peptide length datasets," *Genome Medicine*, vol. 8, no. 1, p. 33, 2016. [Online]. Available: https://doi.org/10.1186/s13073-016-0288-x

[84] R. Vita, J. A. Overton, J. A. Greenbaum, J. Ponomarenko, J. D. Clark, J. R. Cantrell, D. K. Wheeler, J. L. Gabbard, D. Hix, A. Sette, and B. Peters, "The immune epitope database (IEDB) 3.0," *Nucleic Acids Research*, vol. 43, no. D1, pp. D405–D412, 2015. [Online]. Available: https://doi.org/10.1093/nar/gku938

[85] Immune Epitope Database and Analysis Resource, "MHC I Automated Server Benchmarks," http://tools.iedb.org/auto_bench/mhci/weekly/, last accessed 1 Jan 2019.

[86] E. Karosiene, C. Lundegaard, O. Lund, and M. Nielsen, "NetMHCcons: a consensus method for the major histocompatibility complex class I predictions," *Immunogenetics*, vol. 64, no. 3, pp. 177–186, 2012. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/22009319

[87] M. Andreatta, V. I. Jurtz, T. Kaever, A. Sette, B. Peters, and M. Nielsen, “Machine learning reveals a non-canonical mode of peptide binding to mhc class ii molecules,” *Immunology*, vol. 152, no. 2, pp. 255–264, 2017. [Online]. Available: https://doi.org/10.1111/imm.12763

[88] M. Nielsen, C. Lundegaard, T. Blicher, B. Peters, A. Sette, S. Justesen, S. Buus, and O. Lund, “Quantitative predictions of peptide binding to any HLA-DR molecule of known sequence: NetMHCIIpan,” *PLOS Computational Biology*, vol. 4, no. 7, pp. 1–10, 2008. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1000107

[89] W. Fleri, “Selecting thresholds (cut-offs) for MHC class I and II binding predictions,” http://help.iedb.org/entries/23854373, last accessed 1 Jan 2019.

[90] S. Paul, D. Weiskopf, M. A. Angelo, J. Sidney, B. Peters, and A. Sette, “HLA Class I Alleles are associated with peptide-binding repertoires of different size, affinity, and immunogenicity,” *The Journal of Immunology*, vol. 191, no. 12, pp. 5831–5839, 2013. [Online]. Available: https://doi.org/10.4049/jimmunol.1302101

[91] B. Engels, V. H. Engelhard, J. Sidney, A. Sette, D. C. Binder, R. B. Liu, D. M. Kranz, S. C. Meredith, D. A. Rowley, and H. Schreiber, “Relapse or eradication of cancer is predicted by peptide-major histocompatibility complex affinity,” *Cancer Cell*, vol. 23, no. 4, pp. 516–526, 2013. [Online]. Available: https://doi.org/10.1016/j.ccr.2013.03.018

[92] E. Assarsson, J. Sidney, C. Oseroff, V. Pasquetto, H.-H. Bui, N. Frahm, C. Brander, B. Peters, H. Grey, and A. Sette, “A quantitative analysis of the variables affecting the repertoire of T cell specificities recognized after vaccinia virus infection,” *The Journal of Immunology*, vol. 178, no. 12, pp. 7890–7901, 2007. [Online]. Available: https://doi.org/10.4049/jimmunol.178.12.7890

[93] S. H. van der Burg, M. J. Visseren, R. M. Brandt, W. M. Kast, and C. J. Melief, “Immunogenicity of peptides bound to MHC class I molecules depends on the MHC-peptide complex stability,” *J. Immunol.*, vol. 156, no. 9, pp. 3308–3314, 1996. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/8617954

[94] K. W. Jørgensen, M. Rasmussen, S. Buus, and M. Nielsen, “NetMHCstab – predicting stability of peptide–MHC-I complexes; impacts for cytotoxic T lymphocyte epitope discovery,” *Immunology*, vol. 141, no. 1, pp. 18–26, 2014. [Online]. Available: https://doi.org/10.1111/imm.12160

[95] J. J. A. Calis, M. Maybeno, J. A. Greenbaum, D. Weiskopf, A. D. De Silva, A. Sette, C. Keşmir, and B. Peters, “Properties of mhc class i presented peptides that enhance immunogenicity,” *PLOS Computational Biology*, vol. 9, no. 10, pp. 1–13, 10 2013. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1003266

[96] T. Trolle and M. Nielsen, "Nettepi: an integrated method for the prediction of T cell epitopes," *Immunogenetics*, vol. 66, no. 7, pp. 449–456, 2014. [Online]. Available: https://doi.org/10.1007/s00251-014-0779-0

[97] Y. Kim, J. Sidney, S. Buus, A. Sette, M. Nielsen, and B. Peters, "Dataset size and composition impact the reliability of performance benchmarks for peptide-MHC binding predictions," *BMC Bioinformatics*, vol. 15, no. 1, p. 241, 2014. [Online]. Available: https://doi.org/10.1186/1471-2105-15-241

[98] H. Pearson, T. Daouda, D. P. Granados, C. Durette, E. Bonneil, M. Courcelles, A. Rodenbrock, J.-P. Laverdure, C. Côté, S. Mader, S. Lemieux, P. Thibault, and C. Perreault, "MHC class I–associated peptides derive from selective regions of the human genome," *The Journal of Clinical Investigation*, vol. 126, no. 12, pp. 4690–4701, 2016. [Online]. Available: https://doi.org/10.1172/JCI88590

[99] L. R. Olsen, S. Tongchusak, H. Lin, E. L. Reinherz, V. Brusic, and G. L. Zhang, "TANTIGEN: a comprehensive database of tumor T cell antigens," *Cancer Immunology, Immunotherapy*, vol. 66, no. 6, pp. 731–735, 2017. [Online]. Available: https://doi.org/10.1007/s00262-017-1978-y

[100] H. G. Rammensee, J. Bachmann, N. P. N. Emmerich, O. A. Bachor, and S. Stevanović, "SYFPEITHI: database for MHC ligands and peptide motifs," *Immunogenetics*, vol. 50, no. 3, pp. 213–219, 1999. [Online]. Available: https://doi.org/10.1007/s002510050595

[101] N. Vigneron, V. Stroobant, B. J. Van den Eynde, and P. van der Bruggen, "Database of t cell-defined human tumor antigens: the 2013 update," *Cancer Immun*, vol. 13, 2013. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/23882160

[102] H. Thorvaldsdóttir, J. T. Robinson, and J. P. Mesirov, "Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration," *Briefings in Bioinformatics*, vol. 14, no. 2, pp. 178–192, 04 2012. [Online]. Available: https://doi.org/10.1093/bib/bbs017

[103] J. T. Robinson, H. Thorvaldsdóttir, A. M. Wenger, A. Zehir, and J. P. Mesirov, "Variant review with the integrative genomics viewer," *Cancer Research*, vol. 77, no. 21, pp. e31–e34, 2017. [Online]. Available: https://cancerres.aacrjournals.org/content/77/21/e31

[104] A.-M. Bjerregaard, M. Nielsen, S. R. Hadrup, Z. Szallasi, and A. C. Eklund, "MuPeXI: prediction of neo-epitopes from tumor sequencing data," *Cancer Immunology, Immunotherapy*, vol. 66, no. 9, pp. 1123–1130, 2017. [Online]. Available: https://doi.org/10.1007/s00262-017-2001-3

[105] J. Hundal, B. M. Carreno, A. A. Petti, G. P. Linette, O. L. Griffith, E. R. Mardis, and M. Griffith, "pVAC-Seq: A genome-guided in silico approach to identifying tumor

neoantigens," *Genome Medicine*, vol. 8, no. 1, p. 11, 2016. [Online]. Available: https://doi.org/10.1186/s13073-016-0264-5

[106] F. Duan, J. Duitama, S. A. Seesi, C. Ayres, S. Corcelli, A. Pawashe, T. Blanchard, D. McMahon, J. Sidney, A. Sette, B. Baker, I. Mandoiu, and P. Srivastava, "Genomic and bioinformatic profiling of mutational neo-epitopes reveals new rules to predict anti-cancer immunogenicity," *Journal of Experimental Medicine*, vol. 211, no. 11, pp. 2231–2248, 2014. [Online]. Available: http://jem.rupress.org/content/211/11/2231.full

[107] GNU Operating System, "GNU make," https://www.gnu.org/software/make/manual/make.html, last accessed 24 Oct 2019.

[108] "Shark Cluster Homepage," https://git.lumc.nl/shark/SHARK/wikis/home, last accessed 4 Jan 2019.

[109] "Docker Hub Homepage," https://hub.docker.com/, last accessed 5 Jan 2019.

[110] Red Hat, "Search Quay," https://quay.io/search, last accessed 21 Oct 2019.

[111] "Singularity Homepage," https://www.sylabs.io/docs/, last accessed 5 Jan 2019.

[112] "Open Grid Scheduler," http://gridscheduler.sourceforge.net/, last accessed 5 Jan 2019.

[113] Python Software Foundation, "Welcome to Python.org," https://www.python.org/, last accessed 21 Oct 2019.

[114] The R Foundation, "R: The R Project for Statistical Computing," https://www.r-project.org/, last accessed 21 Oct 2019.

[115] Oracle Corporation, "OpenJDK," https://openjdk.java.net/, last accessed 21 Oct 2019.

[116] Oracle, "Java SE at a Glance," https://www.oracle.com/technetwork/java/javase/overview/index.html, last accessed 21 Oct 2019.

[117] Software Freedom Conservancy, "Git," https://git-scm.com/, last accessed 21 Oct 2019.

[118] "Anaconda software distribution. computer software," https://anaconda.com, last accessed 30 Nov 2019.

[119] G. V. der Auwera, "OpenWDL," http://openwdl.org/, last accessed 26 Oct 2019.

[120] "DNAnexus Homepage," https://www.dnanexus.com/, last accessed 5 Jan 2019.

[121] "dxWDL Homepage," https://github.com/dnanexus/dxWDL, last accessed 5 Jan 2019.

[122] J. Organization, “Introducing JSON,” https://www.json.org/, last accessed 27 Oct 2019.

[123] P. Amstutz, M. R. Crusoe, N. Tijanić, B. Chapman, J. Chilton, M. Heuer, A. Kartashov, D. Leehr, H. Ménager, M. Nedeljkovich, and et al., “Common workflow language, v1.0,” 2016. [Online]. Available: https://doi.org/10.6084/m9.figshare.3115156.v2

[124] O. Ben-Kiki, C. Evans, and I. döt Net, “The Official YAML Website,” https://yaml.org/, last accessed 27 Oct 2019.

[125] “BioWDL Homepage,” https://biowdl.github.io/, last accessed 5 Jan 2019.

[126] Open Source Initiative, “MIT License,” https://opensource.org/licenses/MIT, last accessed 27 Oct 2019.

[127] Sequencing Analysis Support Core Team (LUMC), “Biopet Github Repository,” https://github.com/biopet/biopet, last accessed 25 Aug 2020.

[128] Open Source Initiative, “The 3-Clause BSD License,” https://opensource.org/licenses/BSD-3-Clause, last accessed 27 Oct 2019.

[129] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, “Apache spark: A unified engine for big data processing,” *Commun. ACM*, vol. 59, no. 11, p. 56–65, Oct. 2016. [Online]. Available: https://doi.org/10.1145/2934664

[130] “Google cloud homepage,” https://cloud.google.com/, last accessed 18 Nov 2019.

[131] D. Wijnand, “HOCON Informal Specification,” https://github.com/lightbend/config/blob/master/HOCON.md#hocon-human-optimized-config-object-notation, last accessed 29 Oct 2019.

[132] “Singularity Library Homepage,” https://cloud.sylabs.io/library, last accessed 5 Jan 2019.

[133] Red Hat , “Biocontainers,” https://quay.io/organization/biocontainers, last accessed 21 Oct 2019.

[134] D. Cats, R. Vorderman, P. van ’t Hof, and A. Paulo, “BioWDL germline-DNA project, commit 0ec8409a14c718f50130b9af9a1eadd76c28dd09,” https://github.com/biowdl/germline-DNA/tree/0ec8409a14c718f50130b9af9a1eadd76c28dd09, last accessed 27 Oct 2019.

[135] Git, “Git Submodules,” https://git-scm.com/book/en/v2/Git-Tools-Submodules, last accessed 18 Oct 2019.

[136] Object Management Group Unified Modeling Language, “OMG® Unified Modeling Language® Version 2.5.1,” https://www.omg.org/spec/UML/2.5.1, Dec 2017.

[137] P. Ewels, M. Magnusson, S. Lundin, and M. Käller, "MultiQC: summarize analysis results for multiple tools and samples in a single report," *Bioinformatics*, vol. 32, no. 19, pp. 3047–3048, 06 2016. [Online]. Available: https://doi.org/10.1093/bioinformatics/btw354

[138] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with bwa-mem," *arXiv: Genomics*, 2013. [Online]. Available: https://arxiv.org/abs/1303.3997

[139] S. Thankaswamy-Kosalai, P. Sen, and I. Nookaew, "Evaluation and assessment of read-mapping by multiple next-generation sequencing aligners based on genome-wide characteristics," *Genomics*, vol. 109, no. 3, pp. 186 – 191, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0888754317300204

[140] G. A. Van der Auwera, M. O. Carneiro, C. Hartl, R. Poplin, G. del Angel, A. Levy-Moonshine, T. Jordan, K. Shakir, D. Roazen, J. Thibault, E. Banks, K. V. Garimella, D. Altshuler, S. Gabriel, and M. A. DePristo, "From fastq data to high-confidence variant calls: The genome analysis toolkit best practices pipeline," *Current Protocols in Bioinformatics*, vol. 43, no. 1, pp. 11.10.1–11.10.33, 2013. [Online]. Available: https://currentprotocols.onlinelibrary.wiley.com/doi/abs/10.1002/0471250953.bi1110s43

[141] Broad Institute, "(How to) Map reads to a reference with alternate contigs like GRCH38," https://gatk.broadinstitute.org/hc/en-us/articles/360037498992--How-to-Map-reads-to-a-reference-with-alternate-contigs-like-GRCH38, last accessed 13 June 2019.

[142] ——, "How to Call somatic mutations using GATK4 Mutect2," https://gatk.broadinstitute.org/hc/en-us/articles/360035531132--How-to-Call-somatic-mutations-using-GATK4-Mutect2, last accessed 23 Aug 2019.

[143] M. Schirmer, R. D'Amore, U. Z. Ijaz, N. Hall, and C. Quince, "Illumina error profiles: resolving fine-scale variation in metagenomic sequencing data," *BMC bioinformatics*, vol. 17, pp. 125–125, Mar 2016. [Online]. Available: https://doi.org/10.1186/s12859-016-0976-y

[144] Broad Institute, "Mutect2 Notes," https://console.cloud.google.com/storage/browser/gatk-best-practices/somatic-hg38;tab=objects?project=broad-dsde-outreach&prefix=, last accessed 05 Sep 2020.

[145] ——, "GATK Resource Bundle," https://gatk.broadinstitute.org/hc/en-us/articles/360035890811-Resource-bundle, last accessed 05 Sep 2020.

[146] ——, "GATK hg38 Resource Bundle," https://console.cloud.google.com/storage/browser/genomics-public-data/resources/broad/hg38/v0, last accessed 05 Sep 2020.

[147] ——, “GATK Somatic hg38 Resource Bundle,” https://console.cloud.google.com/storage/browser/gatk-best-practices/somatic-hg38, last accessed 05 Sep 2020.

[148] Sequencing Analysis Support Core Team (LUMC), “Chunked-scatter Github Repository,” https://github.com/biowdl/chunked-scatter, last accessed 16 Jul 2019.

[149] AstraZeneca, “VarDict Github Repository,” https://github.com/AstraZeneca-NGS/VarDictJava, last accessed 03 Sep 2019.

[150] Alex Rubinsteyn and Julia Kodysh and B. Arman Aksoy, “Isovar GitHub Repository,” https://github.com/openvax/isovar, last accessed 08 Sep 2020.

[151] “Isovar GitHub Code Frequency,” https://github.com/openvax/isovar/graphs/code-frequency, last accessed 28 Jul 2019.

[152] A. Paulo and D. Ruano, “Binding Prediction Module GitHub Repository,” https://github.com/Amfgcp/NeoSeq_WDL/, last accessed 09 Sep 2020.

[153] Open Source Initiative, “GNU General Public License version 3,” https://opensource.org/licenses/GPL-3.0, last accessed 3 Nov 2019.

[154] A. Rubinsteyn, “PyPi mhctools page,” https://pypi.org/project/mhctools/, last accessed 02 Sep 2019.

[155] A. Paulo and D. Ruano, “Modified mhctools (GitHub fork),” https://github.com/Amfgcp/mhctools, last accessed 09 Sep 2020.

[156] B. Reynisson, B. Alvarez, S. Paul, B. Peters, and M. Nielsen, “NetMHCpan-4.1 and NetMHCIIpan-4.0: improved predictions of MHC antigen presentation by concurrent motif deconvolution and integration of MS MHC eluted ligand data,” *Nucleic Acids Research*, vol. 48, no. W1, pp. W449–W454, 05 2020. [Online]. Available: https://doi.org/10.1093/nar/gkaa379

[157] “Uniprot Homepage,” https://www.uniprot.org/, last accessed 25 Aug 2019.

[158] T. N. M. Schumacher, M. L. H. De Bruijn, L. N. Vernie, W. M. Kast, C. J. M. Melief, J. J. Neefjes, and H. L. Ploegh, “Peptide selection by MHC class I molecules,” *Nature*, vol. 350, no. 6320, pp. 703–706, Apr 1991. [Online]. Available: https://doi.org/10.1038/350703a0

[159] “Biopython Package Documentation,” https://biopython.org/, last accessed 13 Sep 2019.

[160] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman, “Basic local alignment search tool.” *Journal of molecular biology*, vol. 215(3), pp. 403–10, 1990.

[161] C. Camacho, G. Coulouris, V. Avagyan, N. Ma, J. Papadopoulos, K. Bealer, and T. L. Madden, "BLAST+: architecture and applications," *BMC Bioinformatics*, vol. 10, no. 1, p. 421, Dec 2009. [Online]. Available: https://doi.org/10.1186/1471-2105-10-421

[162] Bootstrap Team, "Boostrap - The most popular HTML, CSS, and JS library in the world," https://getbootstrap.com/, last accessed 21 Oct 2019.

[163] S. Frölich, A. Paulo, and D. Ruano, "R Shiny Module GitHub Repository," https://github.com/Amfgcp/neoseq_shiny, last accessed 12 Sep 2020.

[164] A. Paulo, S. Frölich, and D. Ruano, "R Shiny Module Web Deployment," https://neoseq.shinyapps.io/shiny/, last accessed 12 Sep 2020.

[165] RStudio, "Shinnyapps Cloud Service," https://www.shinyapps.io/, last accessed 12 Sep 2020.

[166] A. Paulo and D. Ruano, "R Shiny Module variant count/typing script," https://github.com/Amfgcp/neoseq_shiny/blob/master/shiny/scripts/varCounts.R, last accessed 12 Sep 2020.

[167] "Travis-CI Homepage," https://travis-ci.com/, last accessed 5 Jan 2019.

[168] H. Krekel, "Pytest Framework Homepage," https://docs.pytest.org/en/latest/, last accessed 16 Oct 2019.

[169] A. Paulo, "Merged pull request (#14) containing integration tests," https://github.com/biowdl/somatic-variantcalling/pull/14, last accessed 9 Aug 2019.

[170] P. van 't Hof, R. Vorderman, and D. Cats, "Integrated Pipeline's Travis CI root configuration file," https://github.com/biowdl/germline-DNA/blob/0ec8409a14c718f50130b9af9a1eadd76c28dd09/.travis.yml, last accessed 26 Jun 2019.

[171] N. Karulin, "VarDict's linear memory usage explained (GitHub comment)," https://github.com/AstraZeneca-NGS/VarDictJava/issues/80#issuecomment-357912754, last accessed 27 Mar 2019.

[172] International Cancer Genome Consortium (ICGC) and The Cancer Genome Atlas (TCGA), "ICGC-TCGA DREAM Mutation Calling challenge," https://www.synapse.org/#!Synapse:syn312572/wiki/58893, last accessed 01 Apr 2019.

[173] R. Semeraro, V. Orlandini, and A. Magi, "Xome-blender: A novel cancer genome simulator," *PLOS ONE*, vol. 13, no. 4, pp. 1–19, 04 2018. [Online]. Available: https://doi.org/10.1371/journal.pone.0194472

[174] U. Paila, B. A. Chapman, R. Kirchner, and A. R. Quinlan, "Gemini: Integrative exploration of genetic variation and genome annotations," *PLOS Computational Biology*, vol. 9, no. 7, pp. 1–8, 07 2013. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1003153

[175] S. Foissac, "An integrated encyclopedia of DNA elements in the human genome," *Nature*, 01 2012. [Online]. Available: https://doi.org/10.1038/nature11247

[176] Broad Institute, "Creating HTML Links to IGV," https://software.broadinstitute.org/software/igv/ControlIGV, last accessed 23 Oct 2019.

[177] "Tkinter Homepage," https://docs.python.org/3.4/library/tkinter.html, last accessed 5 Jan 2019.

[178] "Tk Homepage," https://tkdocs.com/tutorial/intro.html, last accessed 5 Jan 2019.

[179] RStudio Inc., "Hosting and deployment," https://shiny.rstudio.com/deploy/, last accessed 24 Oct 2019.

[180] D. Net, "qacct Linux man page," https://linux.die.net/man/1/qacct, last accessed 24 Oct 2019.

[181] "Coveralls Homepage," https://coveralls.io/, last accessed 5 Jan 2019.

[182] Broad Institute, "Picard's LiftoverVcf Tool," https://gatk.broadinstitute.org/hc/en-us/articles/360036831351-LiftoverVcf-Picard, last accessed 27 Sep 2020.

[183] University of California Santa Cruz, "UCSC's Lift Genome Annotations Tool," https://genome.ucsc.edu/cgi-bin/hgLiftOver, last accessed 30 May 2019.

# A

# Cromwell Pipeline Files

## A.1 Cromwell Shark Cluster Configuration

The Cromwell configuration file (in the HOCON format) that was employed in the Shark HPC cluster is shown in Listings A.1-A.13. Together these listings form the whole configuration file. The parameters in this configuration file were defined by the Cromwell configuration administrators of the Shark cluster. The configuration file defines Cromwell's runtime behavior (covered in Section 3.4).

**Listing A.1:** Import of default application values and definition of the Cromwell's server webservice bind port

```
include required(classpath("application"))
webservice {
  port = 8000
}
```

**Listing A.2:** Beginning of the SGE backend-specific configuration

```
backend {
  default="SGE"
  providers {
    SGE {
      actor-factory = "cromwell.backend.impl.sfs.config.
          ConfigBackendLifecycleActorFactory"
```

**Listing A.3:** SGE's general job configuration

```
      config {
        concurrent-job-limit = 200
        exit-code-timeout-seconds = 120
        runtime-attributes= """
          Int? cpu=1
          Int? memory=4
          String? docker
        """
```

**Listing A.4:** SGE's job submission commands for both standard and docker jobs (using SGE's "qsub" command)

```
        submit = """
        qsub \
            -terse \
            -V \
            -b n \
            -wd ${cwd} \
            -N ${job_name} \
            ${true="-pe BWA" false="" defined(cpu)} ${cpu} \
            ${true="-l h_vmem=" false="" defined(memory)}${memory}${true="G"
                false="" defined(memory)} \
            -o ${out} \
            -e ${err} \
            ${script}
        """
        submit-docker = """
        echo '
```

```
33          /usr/local/singularity/3.2.1/bin/singularity exec --containall --bind
                /exports,${cwd}:${docker_cwd} docker://${docker} sh ${script}
34          rc=$?
35          if [ ! -f ${cwd}/execution/rc ]
36          then
37              echo "$rc" > ${cwd}/execution/rc
38          fi
39          ' | \
40          qsub \
41              -terse \
42              -V \
43              -b n \
44              -wd ${cwd} \
45              -N ${job_name} \
46              ${true="-pe BWA" false="" defined(cpu)} ${cpu} \
47              ${true="-l h_vmem=" false="" defined(memory)}${memory}${true="G"
                    false="" defined(memory)} \
48              -o ${cwd}/execution/stdout \
49              -e ${cwd}/execution/stderr
50          """
```

**Listing A.5:** SGE's specification of how to kill a job (*kill* and *kill-docker*), how to check if a job is still running during a cromwell restart (*check-alive*), and how to read a job identifier from the standard output of the submission (*job-id-regex*)

```
51          kill = "qdel ${job_id}"
52          kill-docker = "qdel ${job_id}"
53          check-alive = "qstat -j ${job_id}"
54          job-id-regex = "(\\d+)"
```

**Listing A.6:** SGE's file system duplication strategies when localizing a file (sorted first-to-last). *Cached-copy* helps save space when using docker containers in shared file systems as hard-links do not work between physical disks and soft-links do not work with docker. With cached-copy, files are copied once to the physical disk where the workflow is running and then hard-links are used.

```
55          filesystems {
56            local {
```

```
57              localization: [
58                "soft-link", "hard-link", "cached-copy", "copy"
59              ]
```

**Listing A.7:** SGE's file system duplication strategies when copying a cached file (sorted first-to-last). Also includes file hashing parameters.

```
60              caching {
61                duplication-strategy: [ "soft-link", "hard-link", "copy" ]
62                # Computes an md5 hash of the file path and the last modified
                      time
63                hashing-strategy: "path+modtime"
64                # Checks if a sibling file with the same name and the .md5
                      extension exists, and if it does, uses the content of this
                      file as a hash.
65                check-sibling-md5: true
66              }
67            }
68          }
```

**Listing A.8:** End of SGE backend-specific configuration with number of task retries after transient failures

```
70          default-runtime-attributes {
71            maxRetries: 2
72          }
73        }
74      }
75    }
76  }
```

**Listing A.9:** General system configuration, focused on limiting the number of I/O requests

```
77  system {
78    abort-jobs-on-terminate = true
79    io {
80      number-of-requests = 10000
```

```
    per = 10 seconds
    number-of-attempts = 5
    timeout {
      default = 5 minutes
    }
  }
}
```

**Listing A.10:** Call caching configuration

```
call-caching {
  enabled = true
  invalidate-bad-cache-results = true
}
```

**Listing A.11:** Database configuration

```
database {
  profile = "slick.jdbc.MySQLProfile$"
  db {
    url = "jdbc:mysql://res-crom-db01.researchlumc.nl/cromwell_43_db?useSSL=
        false&rewriteBatchedStatements=true"
    user = "XYZ"
    password = "XYZ"
    driver = "com.mysql.cj.jdbc.Driver"
    # Prevents jobs from failing due to maximum number of connections on the
        mysql server.
    numThreads = 3 # Default was 20.
    maxConnections = 15 # Default
    idleTimeout = "30s" # Default was 10
    maxLifetime = "30m" # Default
  }
}
```

**Listing A.12:** Workflow options

```
workflow-options {
  # Delays failure until all running jobs are complete
  workflow-failure-mode = "ContinueWhilePossible"
}
```

**Listing A.13:** Akka-http [1] (used to serve requests) configuration that limits the load exerted on the HPC node responsible for job submission (head node)

```
# Limit the number of threads on the headnode
akka {
  actor.default-dispatcher.fork-join-executor {
    # Number of threads = min(parallelism-factor * cpus, parallelism-max)
    parallelism-factor = 3.0 # Default
    parallelism-max = 3
    }
}
```

## A.2 User-defined Pipeline Input Files

### A.2.1 Input Definition File

Listings A.14-A.19 show an example of a JSON input file that was used to run the pipeline with real patient data. On the whole, the listings form a single input file (covered in Section 3.7.2.A). The input variables available can be generated with Womtool (covered in Section 3.7.3)

**Listing A.14:** Beginning of the complete user-defined input file example with general parameters

```
{
  "pipeline.outputDir": "/aligning-somatic/results-NIC7-alt-pon",
  "pipeline.dockerTagsFile": "/github/germline-DNA/dockerTags.yml",
  "pipeline.reference": {
    "fasta": "/gatk-b38/Homo_sapiens_assembly38.fasta",
    "fai": "/gatk-b38/Homo_sapiens_assembly38.fasta.fai",
    "dict": "/gatk-b38/Homo_sapiens_assembly38.dict"
  },
```

```
9   "SomaticVariantcalling.runCombineVariants": "true"
```

**Listing A.15:** QC and Adapter Trimming parameters

```
10  "pipeline.dbSNP": {
11    "file": "/gatk-b38/dbsnp_146.hg38.vcf.gz",
12    "index": "/gatk-b38/dbsnp_146.hg38.vcf.gz.tbi"
13  },
14  "pipeline.sample.Sample.library.Library.bqsr.GatkPreprocess.
        baseRecalibrator.knownIndelsSitesVCFs": [
15    "/gatk-b38/Mills_and_1000G_gold_standard.indels.hg38.vcf.gz"
16  ],
17  "pipeline.sample.Sample.library.Library.bqsr.GatkPreprocess.
        baseRecalibrator.knownIndelsSitesVCFIndexes": [
18    "/gatk-b38/Mills_and_1000G_gold_standard.indels.hg38.vcf.gz.tbi"
19  ],
20  "pipeline.sample.Sample.library.Library.readgroup.Readgroup.qc.
        QC.Cutadapt.memory": 16,
```

**Listing A.16:** Read Mapping parameters (the sample configuration file used is shown in Listing A.20)

```
21  "pipeline.sampleConfigFile": "/config/NIC7.yml",
22  "pipeline.bwaIndex": {
23    "fastaFile": "/gatk-b38/Homo_sapiens_assembly38.fasta",
24    "indexFiles": [
25      "/gatk-b38/Homo_sapiens_assembly38.fasta.64.alt",
26      "/gatk-b38/Homo_sapiens_assembly38.fasta.64.amb",
27      "/gatk-b38/Homo_sapiens_assembly38.fasta.64.ann",
28      "/gatk-b38/Homo_sapiens_assembly38.fasta.64.bwt",
29      "/gatk-b38/Homo_sapiens_assembly38.fasta.64.pac",
30      "/gatk-b38/Homo_sapiens_assembly38.fasta.64.sa"
31    ]
32  },
33  "pipeline.sample.Sample.library.Library.readgroup.Readgroup.
```

```
        fastqsplitterR2.memory": 10,
34    "pipeline.sample.Sample.library.Library.readgroup.Readgroup.
        fastqsplitterR1.memory": 10,
```

**Listing A.17:** Somatic Variant Calling parameters – Strelka2

```
35    "pipeline.somaticVariantcalling.SomaticVariantcalling.strelka.
        Strelka.scatterList.scatterSize": 150000000,
36    "pipeline.somaticVariantcalling.SomaticVariantcalling.strelka.
        Strelka.strelkaSomatic.exome": true,
37    "pipeline.somaticVariantcalling.SomaticVariantcalling.strelka.
        Strelka.mantaSomatic.exome": true,
38    "pipeline.somaticVariantcalling.SomaticVariantcalling.strelka.
        Strelka.strelkaGermline.exome": true,
39    "pipeline.somaticVariantcalling.SomaticVariantcalling.strelka.
        Strelka.gatherVariants.memory": 12,
40    "pipeline.somaticVariantcalling.SomaticVariantcalling.strelka.
        Strelka.gatherIndels.memory": 12,
41    "pipeline.somaticVariantcalling.SomaticVariantcalling.strelka.
        Strelka.gatherSVs.memory": 12,
```

**Listing A.18:** Somatic Variant Calling parameters – VarDict

```
42    "pipeline.somaticVariantcalling.SomaticVariantcalling.vardict.
        VarDict.varDict.memory": 18,
43    "pipeline.somaticVariantcalling.SomaticVariantcalling.vardict.
        VarDict.gatherVcfs.memory": 16,
```

**Listing A.19:** Somatic Variant Calling parameters – Mutect2

```
44    "pipeline.somaticVariantcalling.SomaticVariantcalling.mutect2.
        Mutect2.scatterList.scatterSize": 150000000,
45    "pipeline.somaticVariantcalling.SomaticVariantcalling.mutect2.
        Mutect2.scatterList.memory": 10,
```

```
46   "pipeline.somaticVariantcalling.SomaticVariantcalling.mutect2.
        Mutect2.mutect2.memory": 16,
47   "pipeline.somaticVariantcalling.SomaticVariantcalling.mutect2.
        Mutect2.gatherVcfs.memory": 10,
48   "pipeline.somaticVariantcalling.SomaticVariantcalling.mutect2.
        Mutect2.mutect2.germlineResource": "/gm/somatic-hg38_af-only-
        gnomad.hg38.vcf.gz",
49   "pipeline.somaticVariantcalling.SomaticVariantcalling.mutect2.
        Mutect2.mutect2.germlineResource Index": "/gm/somatic-hg38_af-
        only-gnomad.hg38.vcf.gz.tbi",
50   "pipeline.somaticVariantcalling.SomaticVariantcalling.
        variantsForContamination": "/gm/somatic-
        hg38_small_exac_common_3.hg38.vcf.gz",
51   "pipeline.somaticVariantcalling.SomaticVariantcalling.
        variantsForContaminationIndex": "/gm/somatic-
        hg38_small_exac_common_3.hg38.vcf.gz.tbi",
52   "pipeline.somaticVariantcalling.SomaticVariantcalling.
        sitesForContamination": "/gm/somatic-
        hg38_small_exac_common_3.hg38.vcf.gz",
53   "pipeline.somaticVariantcalling.SomaticVariantcalling.
        sitesForContaminationIndex": "/gm/somatic-
        hg38_small_exac_common_3.hg38.vcf.gz.tbi"
54   "pipeline.somaticVariantcalling.SomaticVariantcalling.mutect2.
        Mutect2.mutect2.panelOfNormals": "/aligned-NICs/
        pon_with_gnomAD.vcf.gz",
55   "pipeline.somaticVariantcalling.SomaticVariantcalling.mutect2.
        Mutect2.mutect2.panelOfNormalsIndex": "/aligned-NICs/
        pon_with_gnomAD.vcf.gz.tbi",
56 }
```

### A.2.2 Sample Configuration File

Listing A.20 shows the sample configuration file where the paired tumor-normal FASTQ sample files are specified (covered in Section 3.7.2.B).

**Listing A.20:** Sample configuration file

```
samples:
  - id: NIC7T
    control: NIC7N
    libraries:
      - id: lib1
        readgroups:
          - id: lane1
            reads:
              R1: /data/fastq/NIC7T_L003_R1.fastq.gz
              R2: /data/fastq/NIC7T_L003_R2.fastq.gz
          - id: lane2
            reads:
              R1: /data/fastq/NIC7T_L007_R1.fastq.gz
              R2: /data/fastq/NIC7T_L007_R2.fastq.gz
  - id: NIC7N
    libraries:
      - id: lib1
        readgroups:
          - id: lane1
            reads:
              R1: /data/fastq/normals/NIC7N_L003_R1.fastq.gz
              R2: /data/fastq/normals/NIC7N_L003_R2.fastq.gz
          - id: lane2
            reads:
              R1: /data/fastq/normals/NIC7N_L007_R1.fastq.gz
              R2: /data/fastq/normals/NIC7N_L007_R2.fastq.gz
```

# B

# MultiQC Plots

Figures B.1-B.9 were exported directly from MultiQC. These show plots that MultiQC generated automatically – by analysing the pipeline output directory and its sub-directories – after running the QC and Adapter Trimming and the Read Alignment modules with 9 normal patient samples.

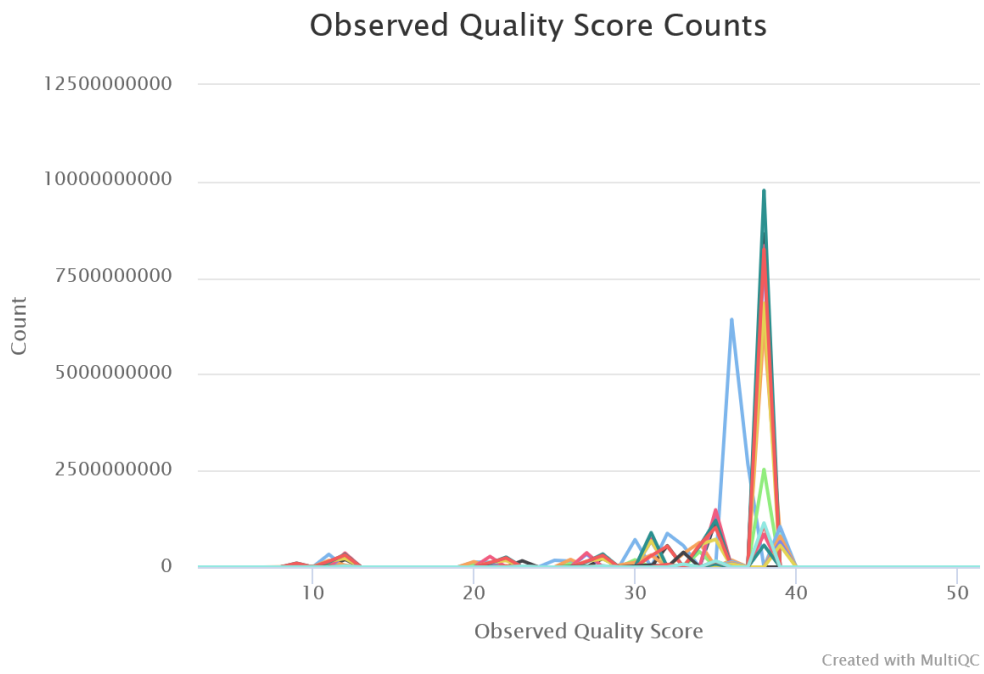


**Figure B.1:** MultiQC's "Observed Quality Score Counts" plot generated from GATK's data.

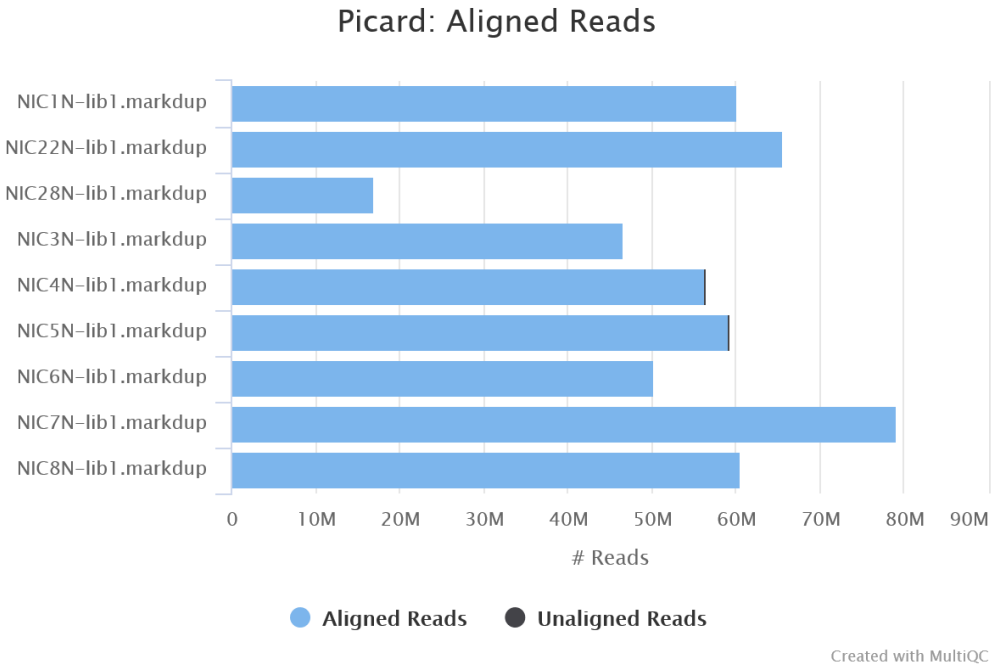


**Figure B.2:** MultiQC's "Alignment Summary" plot generated from Picard's data.

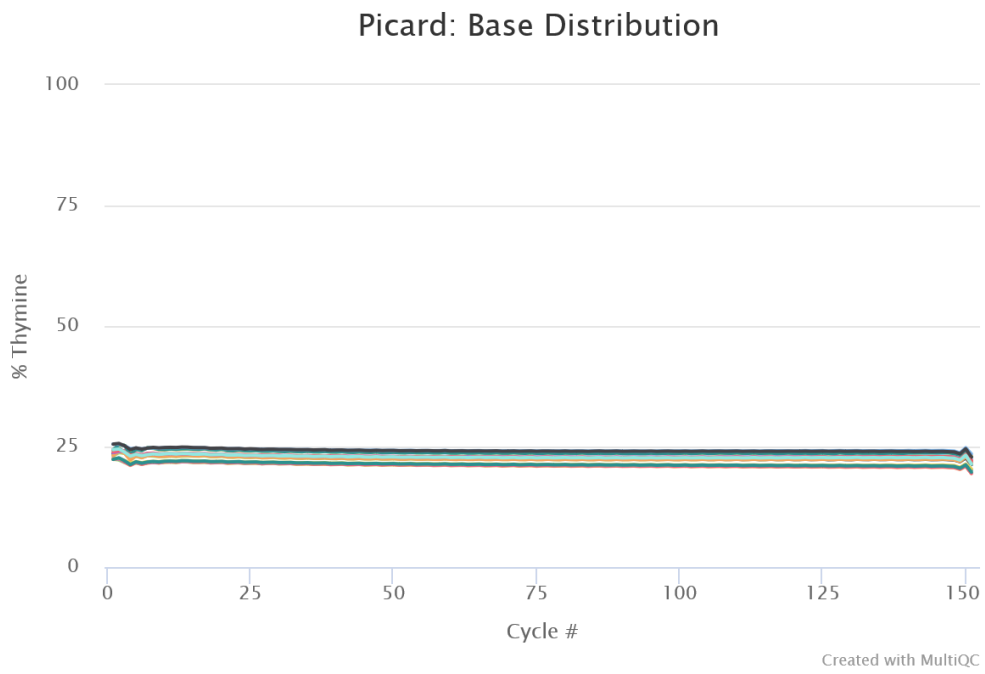


**Figure B.3:** MultiQC's "Base Distribution" plot (Thymine selected) generated from Picard's data.

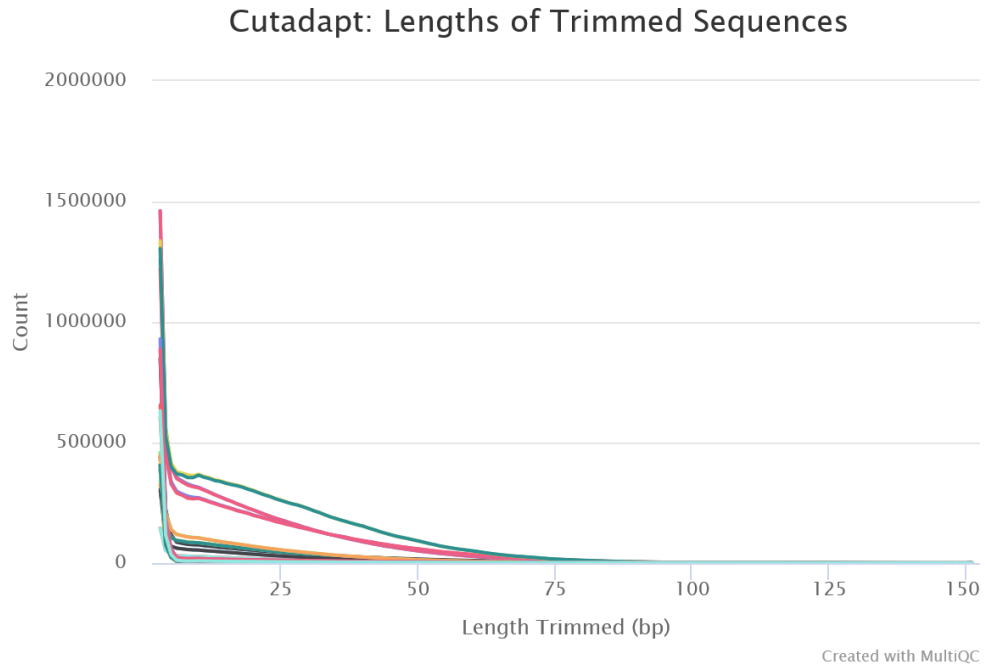


**Figure B.4:** MultiQC's "Lengths of Trimmed Sequences" plot generated from Cutadapt's data.

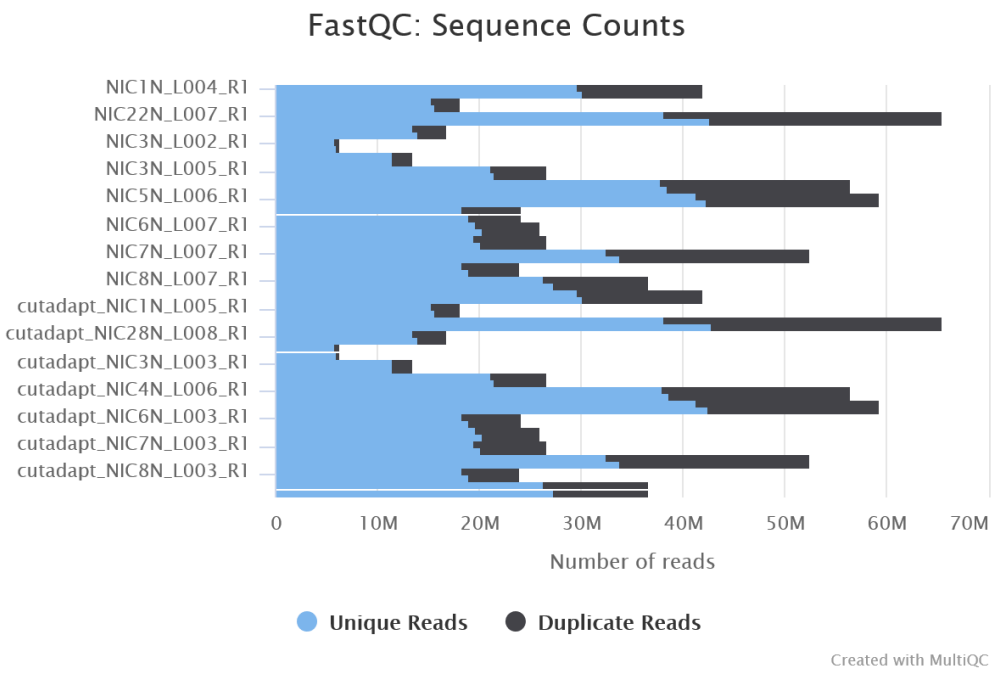


**Figure B.5:** MultiQC's "Sequence Counts" plot generated from FastQC's data.

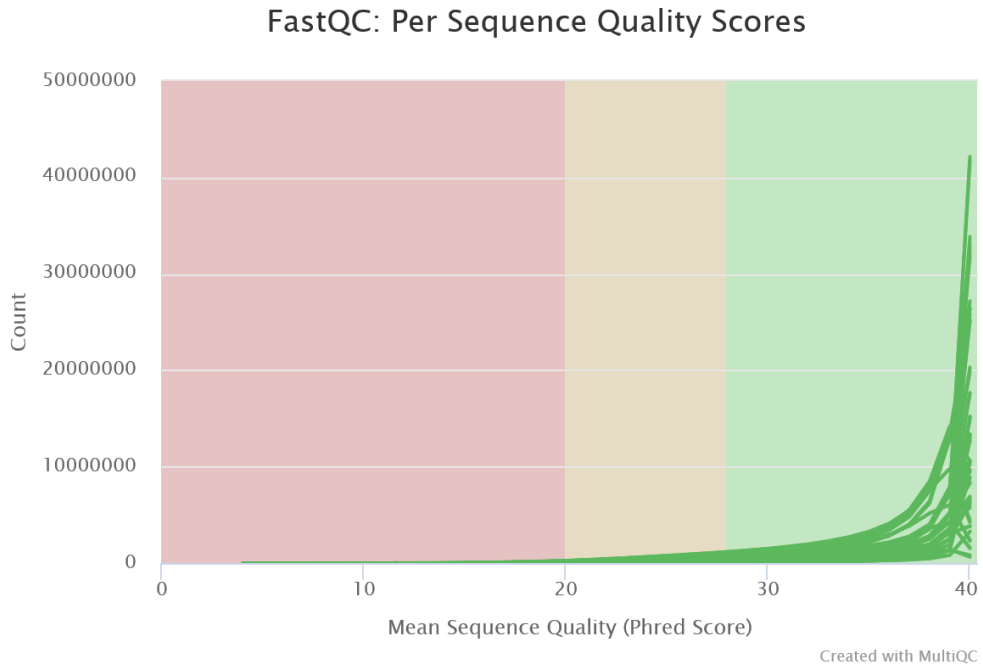


**Figure B.6:** MultiQC's "Per Sequence Quality Scores" plot generated from FastQC's data.

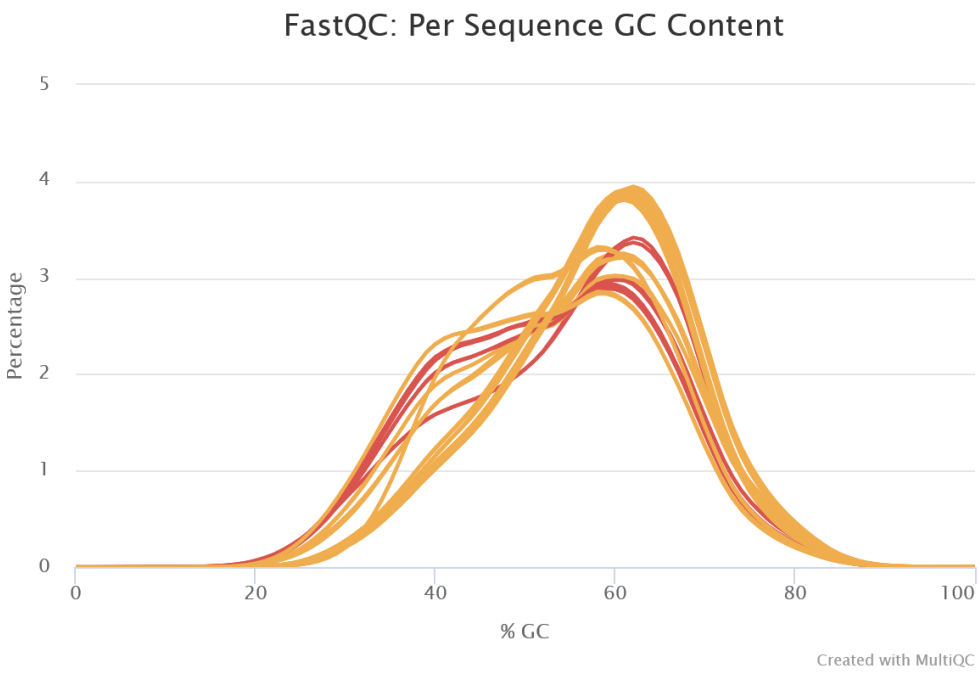


**Figure B.7:** MultiQC's "Per Sequence GC Content" plot generated from FastQC's data.

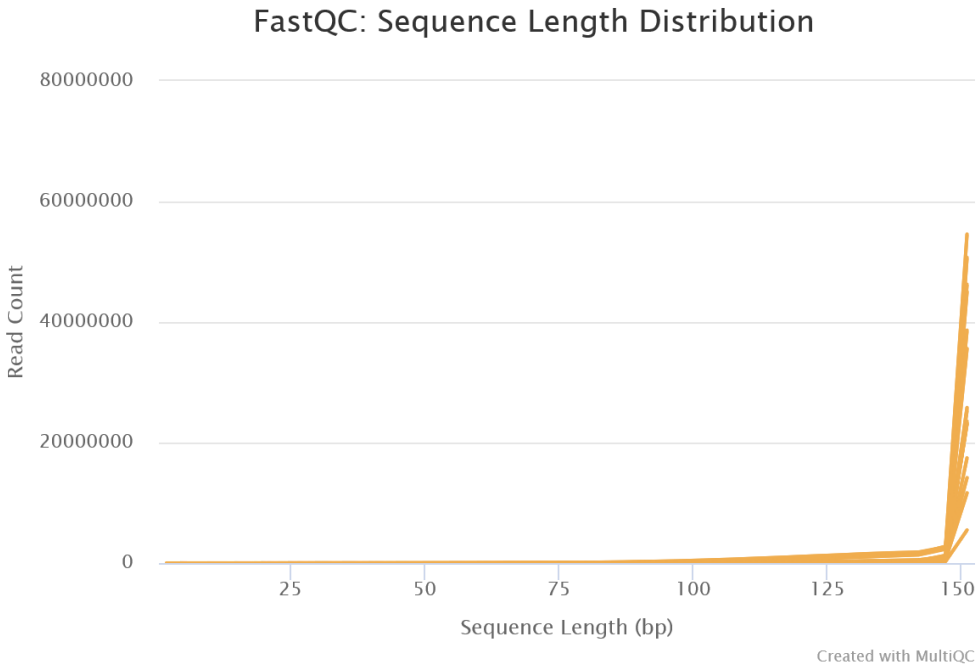


**Figure B.8:** MultiQC's "Sequence Length Distribution" plot generated from FastQC's data.

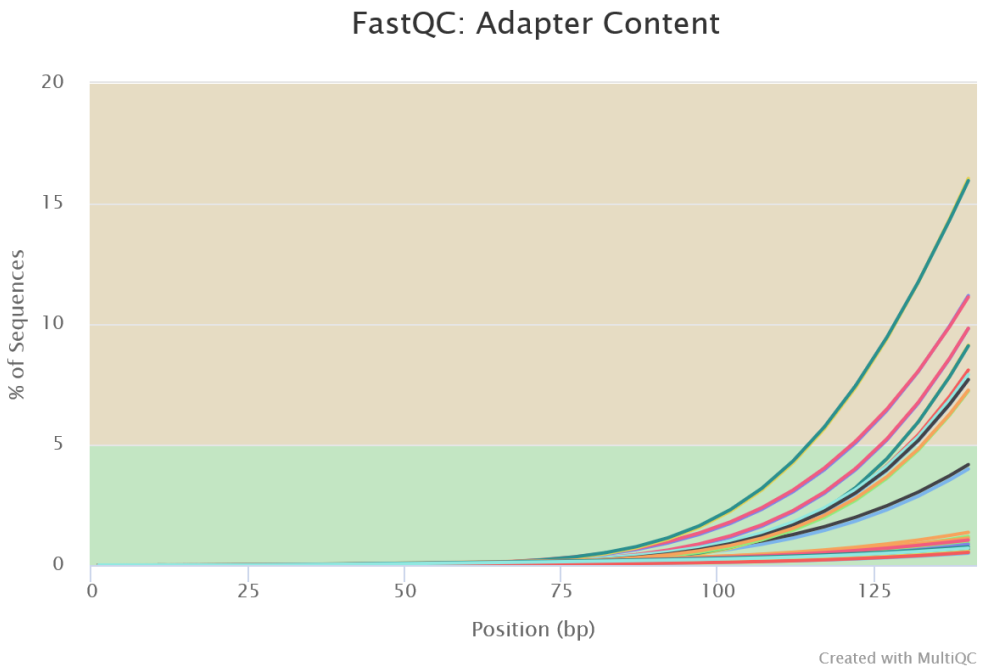


**Figure B.9:** MultiQC's "Adapter Content" plot generated from FastQC's data.

# C

# ICGC-TCGA DREAM Synthetic Dataset Validation

## C.1 Detailed Results

In this appendix, we present the detailed results of Section 4.2.1 resorting to confusion matrices followed by the sensitivity, specificity and precision values. More concretely, the tables shown below detail the summarized results of Figure 4.1 and Table 4.2 (additionally showing precision values). See Section 5.1.2 to understand the disparity between the precision values and the reason for only showing them here. Figure C.1 contains the confusion matrix template for the other tables, where the validation results pertaining to the different somatic variant callers or their union are shown.

| Confusion Matrix Template | | Ground Truth | |
|---|---|---|---|
| | | TRUE | FALSE |
| Predicted | TRUE | TP | FP |
| | FALSE | FN | TN |
| Sensitivity (TPR) | | TP / (TP + FN) | |
| Specificity (TNR) | | TN / (TN + FP) | |
| Precision (PPV) | | TP / (TP + FP) | |

| Mutect2 | | Ground Truth | |
|---|---|---|---|
| | | TRUE | FALSE |
| Predicted | TRUE | 3,443 | 1,170 |
| | FALSE | 94 | 60,787 |
| Sensitivity (TPR) | | 0.973 | |
| Specificity (TNR) | | 0.981 | |
| Precision (PPV) | | 0.746 | |

| Mutect2 and Strelka2 Union | | Ground Truth | |
|---|---|---|---|
| | | TRUE | FALSE |
| Predicted | TRUE | 3,468 | 3,829 |
| | FALSE | 69 | 506,286 |
| Sensitivity (TPR) | | 0.980 | |
| Specificity (TNR) | | 0.992 | |
| Precision (PPV) | | 0.475 | |

| Strelka2 | | Ground Truth | |
|---|---|---|---|
| | | TRUE | FALSE |
| Predicted | TRUE | 3,386 | 2,448 |
| | FALSE | 151 | 469,478 |
| Sensitivity (TPR) | | 0.957 | |
| Specificity (TNR) | | 0.995 | |
| Precision (PPV) | | 0.580 | |

| Mutect2, Strelka2 and VarDict Union | | Ground Truth | |
|---|---|---|---|
| | | TRUE | FALSE |
| Predicted | TRUE | 3,505 | 191,915 |
| | FALSE | 32 | 1,001,961 |
| Sensitivity (TPR) | | 0.991 | |
| Specificity (TNR) | | 0.839 | |
| Precision (PPV) | | 0.018 | |

| VarDict | | Ground Truth | |
|---|---|---|---|
| | | TRUE | FALSE |
| Predicted | TRUE | 3,492 | 189,694 |
| | FALSE | 45 | 716,394 |
| Sensitivity (TPR) | | 0.987 | |
| Specificity (TNR) | | 0.791 | |
| Precision (PPV) | | 0.018 | |

**Figure C.1:** ICGC-TCGA DREAM challenge synthetic dataset detailed validation results. On the top left, the template followed by the remaining tables is shown. The two tables below the template present the results concerning the union of all or part of the somatic variant callers. On the right, the results for each variant caller are shown.

## C.2 Conversion to hg38

Before calling variants on this dataset, we decided to liftover (essentially convert) the ground truth VCF files and re-map the reads, from the human reference hg19, that they came in, to the hg38 reference. We used Picard's and UCSC's tools [182, 183] to lift over the VCFs and we re-mapped the reads as mentioned in Section 3.8.2. This was done because hg38 is the default reference in our pipeline as it greatly expands the repertoire of alternate contigs (contiguous nucleotide sequences) compared to hg19, resulting in more accurate alignments. The liftover resulted in around 17% of the variants in the ground truth file being discarded. These were variants with dated annotations – "INV", "DUP", "DEL", "IGN" and "MSK" – that the liftover tool rejected. For example, we saw "DUP" annotations instead of explicitly stating the duplicated nucleotides. These annotations were located in the "ALT" column of the ground truth VCF file (refer back to the VCF example shown in Listing 2.2). Moreover, these annotations were problematic because the auxiliary tool we use to make VCF comparisons, GATK's CombineVariants, does not recognize these dated annotations, hence being unable to combine the VCF files. Regardless, we considered 17% to be a considerable loss in variants from the ground truth file, so we decided to also call variants using the original files aligned against the hg19 reference and compare them to those obtained with the hg38-aligned files.

After dealing with the three aspects concerning VarDict that were mentioned in Section 4.2.1.A, we were able to run the Somatic Variant Calling module successfully for one of the sets. We analyzed each variant caller's results for each of the two references, hg19 and hg38, to determine which we should use for validation. We will only focus on Mutect2's comparison as the points raised are applicable to the other variant callers. Besides the points in Section 4.2.1.A (only concerning hg19), comparing the results for Mutect2 runs with the hg19 and hg38 references, two aspects arose: 1) with the hg19 reference, we confirm that 17% less variants are called when compared to the ground truth because of dated annotations – "INV", "DUP", "DEL", "IGN" and "MSK". Mutect2 does not annotate variants in such a way. With the hg38 reference, the same 17% of variants had already been discarded following the ground truth VCF liftover, so they were not taken into account. 2) Given that the hg38 reference considers more alternate contigs, Mutect2 called variants located in these alternate contigs that were not in the respective ground truth file but that we considered good predictions (through visual inspection in IGV). Additionally, this made us aware that the ground truth file only considered chromosomes 1 to 22 and X. Aspect "1)" and "2)" were handled similarly as in Section 4.2.1.A: discard variants other than SNVs and discard every alternate contig from our analysis.

Considering the aspects mentioned above, we decided to do the validation using only hg19 related files as there was no benefit in doing otherwise.