POLITECNICO DI TORINO

Master's Degree in Biomedical Engineering

Master's Degree Thesis

FusionFlow: gene fusion detection pipeline in RNA and DNA sequencing data leveraging Nextflow and Docker

Supervisors

Prof. Elisa FICARRA

Ph.D. Marta LOVINO

Candidate

Federica CITARRELLA

December 2021

Abstract

Gene fusion is a phenomenon that occurs when two or more previously independent genes become juxtaposed, forming a single hybrid gene or transcript.
Although several gene fusions detection tools have been developed over the past years, there is still no gold standard for gene fusion discovery. The typical practice is to execute multiple tools and use the union or intersection of their results. This approach is computationally demanding because each tool on its own has specific requirements and generally takes many hours or even days to run.
Several pipelines for gene fusions detection from RNA-seq data were developed. However, analyzing RNA-seq data alone has its limitations. Otherwise, integrating RNA-seq and WGS data leads to a sensitive approach for detecting gene fusion predictions.
The core of this master thesis is the development of a bioinformatic pipeline for the analysis of RNA-seq and WGS data to detect gene fusions. The goal was to realize an easy-to-use, scalable, and highly reproducible pipeline. The proposed workflow includes five gene fusion detection tools: EricScript, Arriba, FusionCatcher, Integrate, and GeneFuse. The main technologies used to build the pipeline are Nextflow, Docker container, and Conda virtual environments to achieve the proposed goal.

Table of Contents

List of Tables

List of Figures

Acronyms

RNA

ribonucleic acid

DNA

deoxyribonucleic acid

WGS

whole-genome sequencing

CML

chronic myelogenous leukemia

ALK

anaplastic lymphoma kinase

FDA

food and drug administration

ALL

acute lymphoblastic leukemia

HPC

high performance computing

FIFO

first in first out

BWA

burrows-wheeler alignment

BLAT

BLAST-like alignment tool

PCR

polymerase chain reaction

BAM

binary alignment map

HDFS

hadoop distributed file system

TSV

tab separate value

VGF

variant call format

OS

operating system

ML

machine learning

Chapter 1

Introduction

1.1 Background

Gene fusion is a phenomenon that occurs when two or more previously independent genes become juxtaposed, forming a single hybrid gene or transcript. This process is the result of deoxyribonucleic acid- or ribonucleic acid-derived rearrangements. Gene fusions remarkably contribute to the evolutionary process by providing a continuous source of new genes. However, at the same time, they often lead to genomic disorders or cancer.
Numerous gene fusions have been recognized as important drivers for a wide array of cancer types. Thus, the discovery of novel gene fusions can better comprehend tumor development and progression. For these reasons, gene fusion detection has become critical to bioinformatics research. [1]

1.2 Gene fusion detection state of art and challenges

Significant advancements have been made in computational methods for fusion gene discovery over the past years due to the widespread applications of Next Generation Sequencing (NGS) technologies. However, bioinformatics analysis of genomic and transcriptomic data is a complex task, and the development of proper bioinformatics tools is necessary. Although many gene fusions detection tools have been developed over the past years, there is still no gold standard. In addition,

the RNA-seq artifacts introduced with library preparation and sequence alignment make it challenging to obtain reliable gene fusions predictions. Therefore, the tools implement rigid filters to maintain the number of false-positive predictions low, causing the side effect that occasionally driver fusions are discarded.
The typical practice is to execute multiple tools and use the union or intersection of their results. Unfortunately, this approach is computationally demanding because each tool on its own has specific requirements and generally takes many hours or even days to run [2]. During the last years, bioinformatics pipelines were developed to deal with these issues. A bioinformatics pipeline is a wide array of algorithms executed in a predefined sequence to process NGS raw sequencing data and generate a list of annotated sequence variants [3]. NGS is the primary technology for discovering gene fusions, and its reducing costs have resulted in an increasing quantity of patients with whole transcriptome sequencing (RNA-seq) and whole-genome sequencing (WGS) data.
Several pipelines for gene fusions detection from RNA-seq data have been developed. However, analyzing RNA-seq alone has its limitations. Otherwise, integrating RNA-seq and WGS data leads to a sensitive approach for detecting gene fusion predictions [4]. Indeed, being generated separately, WGS and RNA-seq data do not present the same artifacts and noise, and their integration can result in better analysis. Furthermore, this process makes the prioritization of biologically relevant gene fusion easier, often masked by false positives.

1.3 Thesis outline

The core of this master thesis is the development of a bioinformatics pipeline for the analysis of RNA-seq and WGS data to detect gene fusions. The goal was to realize an easy-to-use, scalable, and highly reproducible pipeline. The proposed pipeline includes five gene fusion detection tools: EricScript, Arriba, FusionCatcher, Integrate, and GeneFuse. In order to achieve the proposed goal, the leading technologies used to build the pipeline are Nextflow, Docker container, and Conda virtual environments. Nextflow was used to obtain an easily reproducible and scalable pipeline. It permits the creation of complex processes running sequentially or concurrently, giving the possibility to install the tools directly inside the pipeline before execution

and, at the same time, to parallelize the different tools in the interest of time. Docker container technology was introduced to run operations inside the pipeline quickly and reliably. Finally, Conda was exploited to create environments and easily switch between them, allowing flexible environment management.

Chapter 2

Gene Fusions

2.1 Definition

The expression *gene fusion* refers to hybrid genes formed when two or more previously independent genes become juxtaposed. It can occur due to different biological phenomena such as translocation, interstitial deletion, or chromosomal inversion. A fusion can be found at the DNA level or RNA level. Gene fusions transcription is not the only cause of chimeric RNA origination: they can also derive from trans-splicing of two pre-mRNAs and alternative splicing of a read-through transcript, known as cis-splicing of adjacent genes.

Among the most widely studied genomic variations, gene fusions have been of great interest due to their correlations with tumorigenesis. A canonical example of fusion genes is BCR-ABL, which is translated into an abnormal tyrosine kinase that leads to the development of chronic myelogenous leukemia (CML). A BCR-ABL targeting drug, Gleevec/Glivec, was very successful in the treatment of CML, prompting the search for other fusion genes to be used as tumor-specific biomarkers or drug targets [5].

Gene fusions oncogenic properties are expressed by either forming a hybrid protein with oncogenic functionalities (e.g., by causing the activation of a specific enzyme), deregulating one of the implicated genes (e.g., by fusing a strong promoter to a proto-oncogene), or inducing a loss of function (e.g., by truncating a tumor suppressor gene). One estimate states that translocations and gene fusions are responsible for 20% of global cancer morbidity.

The prevalence of gene fusions differs extensively between different cancer types: gene fusions cause 90% of all lymphomas, more than 50% of leukemias, and 30% of soft tissue tumors. For example, in prostate cancer, the most common genetic alteration is the TMPRSS2-ERG, being found in over 50% of patients, but many recurrent gene fusions occur at low frequencies, such as the KIF5B-RET fusion, which is present in less than 3% of lung adenocarcinomas.
Gene fusions have also been important in classifying molecular subtypes of cancers, monitoring enduring disease post-treatment, and predicting eventual relapse. Fusion transcripts are also emerging as therapeutic targets. For example, a significant patient outcome improvement was followed by drug development targeting the ATP-binding sites and allosteric regions of the BCR-ABL fusion kinase, an active tyrosine kinase, and the driving mutation in chronic myelogenous leukemia. Another example is shown by the inhibitors of the anaplastic lymphoma kinase (ALK) protein; they have greatly improved prospects for patients with EML4-ALK fusion positive non-small cell lung tumors [6], [1].

2.2 Mechanisms involved in the formation of gene fusions

The generation of a chimeric gene generally occurs in two phases. The partial (or complete) sequences that will eventually compose the new single hybrid gene must be placed in proximity on a chromosome unless the fusion is generated by trans-splicing or occurs between two genes already adjacent to each other. Although this step alone may form a new gene structure with a chimeric transcript, a second step is often required: it may involve deletions, acquisition of new splice sites, and/or engagement of non-coding regions. Importantly, duplicated gene copies are often involved in the formation of new genes because, in this case, new chimeric proteins can be generated without the disruption of the original gene functions [1].

Chromosome structural variations may result in the exchange of coding or regulatory DNA sequences between genes. The chromosome rearrangements that lead to gene fusions can be balanced and unbalanced. Balanced rearrangements include insertions (in which a chromosomal portion is moved to a new interstitial

position in the same or another chromosome), inversions (which involve rotation of a chromosomal portion by 180°), and balanced translocations (which involve the transfer of chromosomal segments between chromosomes). Unbalanced rearrangements include deletions (characterized by the deficiency of a chromosomal segment), duplications (in which a chromosome portion repeats within the same chromosome), and unbalanced translocations (in which chromosomal segments are transferred between chromosomes to cause partial monosomy and partial trisomy)(Figure 2.1)[7].

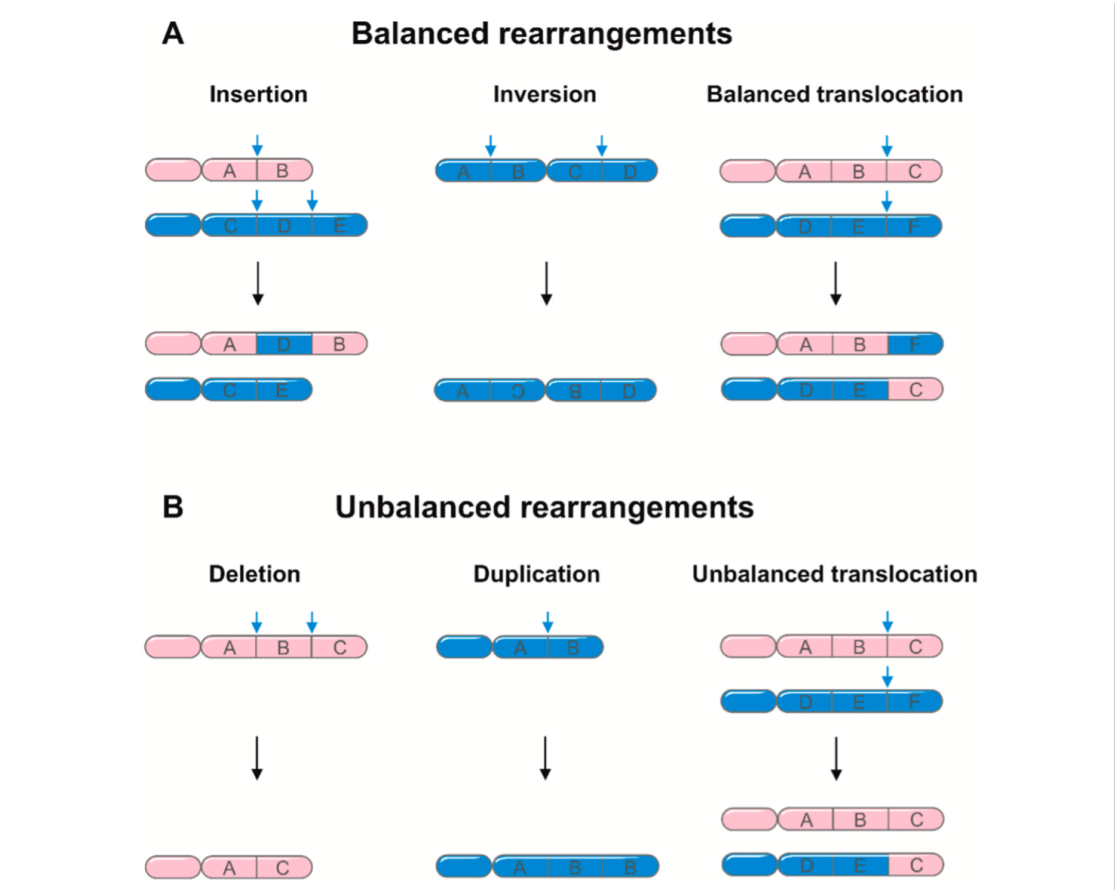

Figure 2.1: Balanced (A) and unbalanced (B) chromosome rearrangements.

2.3 Gene fusion detection and applications in cancer

Gene fusions play significant roles in the first steps of tumorigenesis. The first gene fusions were originally identified in the early 1980s in B cell lymphomas. At that time, gene fusions mainly referred to leukemia and sarcomas, in which single-nucleotide variants burden is limited. These cancers contain conserved gene fusions, some of which were found in almost 100% of cancer subtypes. The development of research on NGS drives a vast number of new gene fusions discovery in all kinds of neoplasia. In 2018, Gao used STAR-Fusion, EricScript, and Breakfast as detection tools to identify 25,664 fusions based on RNA-seq data from 9624 tumors. Many gene fusions are solid driver mutations in neoplasia and have provided fundamental insights into tumorigenesis.

The tight association between specific gene fusions and their related tumor phenotypes makes gene fusions ideal for (a) diagnostic purposes, such as BCR-ABL1 fusion protein derived from Philadelphia chromosome 9–22 translocation in chronic myeloid leukemia (CML) and EWSR1-FLI1 in Ewing's sarcoma; (b) risk stratification, such as findings that fusions strongly stratified patients of prostate cancer across low and intermediate/high-risk groups and levels of TMPRSS2/ERG fusion transcript in urine can be used to stratify prostate cancer risk in men with elevated serum PSA; (c) targeted drugs. Nowadays, gene fusions are essential for treating cancer patients, as many drugs selectively target the proteins encoded by these genes. Imatinib, a tyrosine kinase inhibitor accepted by the US Food and Drug Administration (FDA) in 2001, was the first drug designed to target BCR-ABL1 fusion protein in CML. Imatinib significantly improved CML patients' lifespan and quality of life and effectively treated other tumors. After the success of Imatinib, several new drugs have been realized to treat cancers with fusion genes. For example, Dasatinib, Nilotinib, Bosutinib, and Ponatinib have been approved to treat CML or acute lymphoblastic leukemia (ALL) with the BRC-ABL1 fusion protein. Crizotinib, Ceritinib, Alectinib, Brigatinib, Lorlatinib, the first-, second-, third-generation ALK inhibitors, have been approved to treat NSCLC with ALK fusions. The first-generation TRK inhibitors, Larotrectinib and Entrectinib, were approved to treat various adult and pediatric cancers with TRK fusion in 2018 and

2019.

In general, tumors with gene fusions show a high level of malignancy and progress rapidly. Although targeted drugs work well for some fusions, no functional therapies exist for many tumors with gene fusions. For this reason, it is necessary to develop new therapies to treat tumors with gene fusions. Recently, tumor neoantigens immunotherapy has become a promising cancer treatment. Furthermore, in most types of cancer, fusion neoantigens have higher immunogenic potential than SNV and Indel (SNVIndel) neoantigens, suggesting that gene fusions may be suitable targets for cancer immunotherapy [7].

Chapter 3

Materials and Methods

3.1 Materials

Executing gene fusion detection tools is a computationally demanding task. Thus, high computer performances are required to run these tools.
For this thesis, Hactar cluster of HPC@POLITO Academic Computing Center and POLITO Philae sever were used. The HPC Politecnico di Torino project is an Academic Computing center that provides computing resources and technical support for academic research activities. Philae is a POLITO internal development server. The disk quotas of the servers were adapted to make the system compliant with the necessary resources. Hactar Cluster and Philae server's technical specifications are shown in Table 3.1.

	HACTAR	**PHILAE**
OS	CentOS 7.6 - OpenHPC 1.3.8.1	Ubuntu Linux 18.04
RAM	3.7 TB	131 GB
CPU	2x Xeon E5-2680 v3 2.50 GHz (turbo 3.3 GHz) 12 cores	Intel Xeon E5-2630 v3 2.40 GHz 8 cores

Table 3.1: Server's technical specifications.

Visual Studio Code was used to realize the Nextflow code, the Dockerfile, the YAML Conda environment files and to interact with the remote repository.
GitHub and DockerHub were used as hosted repository to store and exchange files

related to the pipeline.

3.2 Methods

The approach used for the realization of the pipeline is based on four main steps:

1. **analysis**;
2. **design**;
3. **development**;
4. **validation**.

The analysis step represents the preliminary stage in which several investigations are performed. In this first step, the pipeline requirements are defined, and the gene fusion tools are firstly collected and then analyzed through their documentation. Subsequently, the design step occurs. In this phase, a theoretical model of the pipeline is realized. The theoretical model is built considering the requirements defined in the previous step. The third step is the development of the code. In this stage, the pipeline is realized according to the model early defined. The last step regards the validation of the pipeline. In this phase, several tests are performed in order to verify that the pipeline works correctly.
These steps were iteratively repeated several times to obtain the most functional model. The pipeline development life-cycle is shown in Figure 3.1.

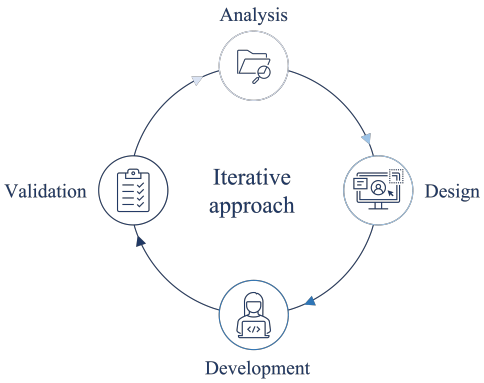

Figure 3.1: Iterative approach steps.

Chapter 4

Pipeline

4.1 Introduction

The discovery of novel gene fusions can lead to a better comprehension of tumor progression and development. However, detecting them through gene fusion tools is quite a challenging task. There are several limitations in traditional tools usage:

- each tool has specific installation requirements and version dependencies that must be precisely adhered to;
- files and databases downloads and tools execution are time-consuming;
- distinct tools can require different input data formats;
- multiple complementary fusion detection tools are needed to improve sensitivity.

Bioinformatics algorithms and processes executed in a specific sequence to process NGS data are collectively referred to as the NGS bioinformatics pipeline. A bioinformatics pipeline processes numerous sequence data and their associated metadata through multiple transformations using a series of software components, databases, and operation environments (hardware and operating system) [5].

FusionFlow is an easily reproducible and scalable bioinformatics pipeline for detecting gene fusions from both RNA-seq and WGS data. It includes five fusion detection tools executed through fifteen processes. The processes are built using Nextflow

and exploit Docker and Conda technologies. Docker and Conda engines are used to create virtual environments precisely configured for each tool. In addition, Nextflow allows running tools downloads, installation, and execution concurrently in the interest of time constraints. Finally, the pipeline inputs standard data formats and eventually converts them directly inside specific converter processes.

4.2 Technologies

In order to realize an easy-to-use, scalable, and highly reproducible pipeline, the main technologies used to build the pipeline are Nextflow, Docker container, and Conda virtual environments.

4.2.1 Nextflow

Using Bash scripts would be the simplest way to create a processing pipeline, but this approach is limited. A Bash script does not parallelize to multiple subjects and is challenging to maintain over time. Numerous pipelining tools exist and allow to develop a pipeline more easily than with Bash script, such as Luigi [8] or Nextflow [9]. Generally, Luigi pipelines creation and maintenance are more complex than a Nextflow pipeline [10]. FusionFlow is built using Nextflow. Nextflow is an open-source software used to write, deploy and share workflows in a very portable manner. Its Domain Specific Language (DSL) enables the development and implementation of parallel workflows on clusters and clouds. Thus, Nextflow enables scalable and reproducible scientific workflows wrapping every task of the workflow inside a container. Indeed, Nextflow supports the usage of Docker and Singularity containers. The main Nextflow components are processes and channels.

Prosesses

In Nextflow, a process is the basic element to execute a user script. A process statement begins with the keyword "process", followed by the chosen process name, and lastly, the process body is enclosed in curly brackets. The process body must include a script that is executed by it. The *script* block is a string declaration that includes the command that is executed by the process. A process presents just one *script* block, and it must be the last declaration when the process also contains

other declarations such as input and output. The entered string is executed in the host system as Bash script by default. It can be a command, a script, or a combination of them, generally used in a terminal shell or a Bash script. The *script* block can include just a simple string or multi-line string. Nextflow strings can be enclosed in single-quotes or double-quotes, and multi-line strings are defined by three single-quote or three double-quote characters. Strings enclosed in double-quotes support variable substitutions, while strings delimited by single-quotes do not.

The *script* block does not allow the direct usage of both Nextflow and Bash variables. To escape this problem *shell* block can be used, instead of the common *script* block. The *shell* block is a string declaration that describes the shell command executed by the process. Differently from the script definition, it uses the exclamation mark ! character as a placeholder for Nextflow variables instead of the usual dollar $ symbol. In this way, it is possible to use both Nextflow and Bash variables in the same part of code, maintaining process scripts readable and easy to fix.

Nextflow processes are isolated from each other. Communication between channels is possible through Nextflow channels. Inside the process definition, the input block indicates from which channels the process expects to receive data. It can be defined as one *input* block at a time, and it must contain one or more input declarations. An input declaration includes an input qualifier and the input name, then it is followed by the keyword "from" and the channel name over which inputs are expected. Finally, to enable specific functionalities, other input optional attributes can be specified. The input qualifier defines the type of data to be received. Nextflow uses this information in order to apply the semantic rules associated with qualifiers and adequately handle them.

The "val" qualifier allows the handling of data of any type as input. The "file" qualifier allows receiving file values in the process execution context. In this case, Nextflow stages the file in the process execution directory allowing its usage in the script by calling the name in the input declaration. The "tuple" qualifier allows handling multiple parameters in a single parameter definition. It can be useful when a process receives input values that need to be managed separately. Each element in the tuple definition is associated with a corresponding element with the tuple definition.

The *output* declaration block allows specifying the channels used by the process to send out the results obtained. One *output* block at a time can be defined, and it must contain one or more output declarations. Output definitions start with an output qualifier and the output name, followed by the keyword "into" and one or more channels over which outputs are sent. Finally, to enable specific functionalities, other input optional attributes can be specified.
If the keyword "val" is used as a qualifier, then a value defined in the script context is given as output. The "file" qualifier is used, then one or more files produced by the process are given as output over the specified channel. The "tuple" qualifier allows sending multiple values into a single channel. This feature is useful when there is the needing to collect together the results of multiple executions of the same process.
The *when* declaration allows to specify a condition that must be verified in order to execute the process. This aspect can be any variable that evaluates a boolean value. Generally, it is useful to enable the process execution according to the state of specific inputs and parameters.
The standard process syntax is shown in Figure 4.1. [9]

```
process < name > {

    [ directives ]

    input:
    < process inputs >
    output:
    < process outputs >
    when:
    < condition >

    [script|shell|exec]:
    < user script to be executed >
}
```

Figure 4.1: Standard process syntax.

Channels

Nextflow uses the Dataflow programming model in which processes communicate through channels. A channel has two principal properties:

1. sending a message is an asynchronous operation that happens immediately, without having to wait for the receiving process;

2. receiving data is a blocking operation that stops the receiving process until the data have arrived.

Nextflow provides two different types of channels: *queue channels* and *value channels*.
A *queue channel* is a non-blocking FIFO queue that unidirectionally connects two processes or operators. It is generally created using methods such as a from or fromPath. A queue channel can also be chained with channel operators such as map or flatMap. This type of channel is also generated by process output declarations using the into clause.
A *value channel* by definition is bound to a single value, and it can be read unlimited times without consuming its content. A value channel is created using value methods or by operators returning a single value, such as first, last, or collect. [9]

4.2.2 Docker

Docker is a platform that allows developers and systems operators to develop, deploy, and execute virtualized applications in sandbox environments known as containers. Containers are instances of an image, abstractions at the app layer that wraps code and dependencies together, so the application runs quickly and reliably from one computing environment to another. An image contains all the components necessary to execute the application: a) the application itself, b) the libraries and packages needed, c) the environment variables, and d) the configuration files. The underlying technology behind Docker is a sandbox environment that virtualizes the underlying host without the need for a total revision of the application (Figure 4.2).

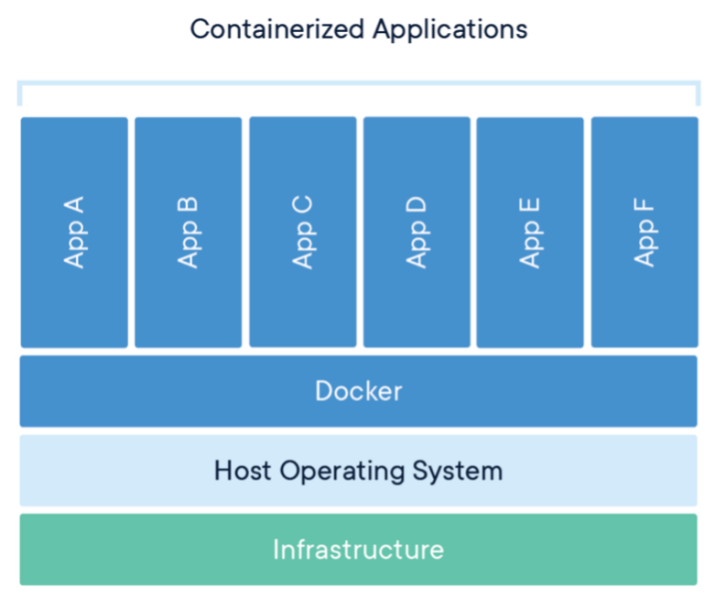

Figure 4.2: Docker layers.

This technology is mainly designed for security and isolation from the host. The main properties that make containers a leader in virtualization technologies are [11] [12]:

- **Flexibility**: docker technology allows to virtualize and instantiate lightweight applications that use only a few libraries up to heavy applications with images reaching GB in terms of size.
- **Lightweight**: containers share the host kernel. This step allows them to excel in lightweight application provisioning. It also reduces the size and processing times.
- **Interchangeable**: docker flexibility enables developers to deploy program updates and upgrades quickly. While the application itself is running, its newer version is deployed. Subsequently, the newer version takes over the tasks of the previous one.
- **Portable**: docker technology provides a sandbox environment that allows to build applications locally and upload to the cloud, download and execute

them anywhere.

- **Scalable**: the increase and decrease of instances can be automated, so the number of replicas can be dynamically adjusted to the requirements of users.

- **Stackable**: Docker enables the possibility to create a stack to share dependencies that can be orchestrated together. This property wraps services together in order to save time, resources, and container size. This stacking can also be performed while the application runs.

It is required a file known as Dockerfile to create a Docker image. The Dockerfile contains all information on how to build the image. The Dockerfile used to build the image for FusionFlow is shown in Appendix A.
The image is stored and publicly available on the DockerHub repository (https://hub.docker.com/layers/177825516/federicacitarrella/pipeline/latest/images/sha256-fc28abaa76c86772dd92088a93a6c0652fc35a0c926cc234b8698d240258a040?context=repo).

4.2.3 Conda

Tools generally require different software packages and dependencies. Setting up an environment and installing all dependencies can be long, tedious, and difficult, especially for beginners and non-technical experts. Several users configure their environment without specifying the package version number. Some tools such as virtual environments exist to realize several working environments with proper package versioning. However, this multi-environment concept can be difficult for most non-computer-scientist users. Therefore, the proposed pipeline includes the creation of virtual environments with specific versions for each tool, making the execution easy for everyone.
To correctly manage software packages and dependencies, Conda technology is used. Conda is a flexible open-source package management and environmental management system. It easily creates and switches between environments, allowing flexible management of environments.

For the pipeline, five Conda virtual environments were created. These environments

were defined through five yml files. Yml files include three sections: a) name, b) channels and c) dependences. The first section defines the name of the environment. The second specifies the channels where packages are stored and the last one defines the package to be installed and their specific version. Table 4.1 are shown the packages installed in each environment and their version.

Environment	**Dependencies**	**Versions**
ericscript		
	ericscript	0.5.5
	gdown	3.13.0
arriba		
	arriba	2.1.0
fusioncatcher		
	fusioncatcher	1.33
genefuse		
	genefuse	0.6.1
	samtools	1.13
	gdown	3.13.0
integrate		
	cmake	3.18.2
	parallel	20210822
	boost	1.76.0
	samtools	1.14
	bowtie2	2.4.4
	bwa	0.7.17
	gdown	3.13.0

Table 4.1: Environments dependencies and versioning.

4.3 Tools

FusionFlow includes five fusion detection tools: EricScript, Arriba, FusionCatcher, GeneFuse and Integrate. Three of them, EricScript, Arriba, and FusionCatcher, accept as input just RNA-seq data, GeneFuse accepts just WGS data, and Integrate can accept as input just RNA data or both RNA and WGS data.

- **Arriba**: is a linear workflow with just a single alignment process followed

by a filtering step. Arriba is based on the ultrafast STAR RNA-seq aligner [13]. STAR can search for two types of chimeric alignments: split reads and spanning reads. The chimeric alignments are stored in a separate output file or in the main output file. Arriba analyzes the chimeric alignments from either of these files and finds gene fusions. STAR aligns reads that support these fusions as if the fused genes were obtained by splicing because STAR defines the type of alignment just by analyzing the dimension of the gap rather than the gene annotation. Moreover, to avoid fusions resulting from focal deletions, Arriba screens for alignments spanning the annotated genes' edges. Unlike many other fusion detection pipelines, Arriba can reuse already existing alignments of STAR rather than align reads exclusively to call gene fusions. Once the candidate alignments have been stored, Arriba utilizes a set of filters to reach high-confidence predictions and remove artifacts. [2]

- **Ericscript**: a computational framework that identifies fusion transcript signatures using a combination of four alignment processes. It comprises the following steps: (1) mapping of the reads to the transcriptome, (2) finding of discordant alignments and building of the exon junction reference (3) recalibration of the exon junction reference, (4) scoring and filtering the candidate gene fusions. The first alignment is used to find discordant alignments and to build an exon junction reference, it is performed by Burrows–Wheeler Alignment (BWA) [14]. This step precedes the mapping of all the reads to this novel reference to discover reads that are not properly mapped. To accurately estimate the junctions positions, EricScript performs a further local alignment of the not properly mapped reads against the exon junction reference using BLAT [15]. The last mapping step allows for the detection of the spanning reads, that is, the reads that span across the junctions and generate a list of candidate fusions. Finally, EricScript calculates a probability score for each predicted fusion and removes analysis artifacts using several heuristic filters. [16]

- **FusionCatcher**: a software tool for finding somatic fusion that achieves competitive detection rates and real-time PCR validation rates in RNA-seq data from tumor cells. Firstly, some pre-processing and filtering are performed

on the input data. Quality filtering is performed by four steps: (1) removing unnecessary reads, (2) trimming the reads which contain adapters and poly-A/C/G/T tails, (3) clipping the reads valuating quality scores, (4) removing the reads which are considered as bad quality by Illumina sequencer. FusionCatcher performs most of the data analysis at the RNA level by aligning the reads on transcriptome using Ensembl genome annotation [17], and Bowtie aligner [18]. The reads that show a better alignment, with just a few mismatches, at the genome level will have their transcriptome mappings removed. Likewise, the reads which map simultaneously on several transcripts of different genes have their mappings removed. The unmapped reads, which are the reads that succeed the quality filtering and do not map on the transcriptome and the genome, are kept for additional analyses. The reads mapping on the transcriptome are collected and used further to create the first list of candidate fusion. Pairs of genes are removed from the preliminary list of candidate fusion genes using specific biological criteria. FusionCatcher approach consists of four different methods and aligners for detecting the fusion junctions. The aligners are Bowtie [18], BLAT [15], STAR [13], and Bowtie2 [19], each method corresponds to one of them. [20]

- **GeneFuse**: a tool to detect and visualize target gene fusions by analyzing DNA-seq data. GeneFuse is based on four major steps: (1) indexing, (2) matching, (3) filtering, and (4) reporting. Firstly, GeneFuse extracts the sequences from the reference genome within the fusion gene areas. A k-mer of all these sequences is computed, and each element of the k-mer is associated with a list of genome coordinates that it matches. A hashmap is produced, and it is used for mapping a read to the target genes. In the matching step, a set of sequences is computed for each read by storing all its subsequences with a length of k. Then the associated genes of this read can be discovered by mapping the subsequences to the genome coordinate using the index computed in the previous step. When the fusion match prediction list is created, a newer k-mer is formed by enumerating all subsequences of the reads supporting the fusions. Then the entire reference genome is analyzed for searching the same k-mer elements, and the matched genome coordinates will similarly be

collected to prepare a new global index G. For each read in the fusion match candidate list, it is mapped to G, and if it could be mapped to a reference genome, it is removed from the fusion match candidate list. Other filters can also be applied to eliminate false callings. In the reporting step, the fusion matches are firstly clustered as fusion results respect the fusion points. Each fusion result is further analyzed and subjected to other filters and the ones passing these are considered qualified fusions, for which the exon or intron of the fusion breakpoint will be located. [21]

- **INTEGRATE**: a tool finding gene fusions with exact fusion junctions and genomic breakpoints using RNA-seq and WGS paired-end sequencing data aligned to the reference genome in BAM format. INTEGRATE is a flexible tool admitting to use reads aligned by different tools, including GSNAP [22], TopHat2 [23], and STAR [13]. Considering that gene fusions expression is relevant, INTEGRATE primarily analyzes mapped and unmapped RNA-seq reads, then analyzes WGS reads from tumor and a normal sample, if available. INTEGRATE is based on the following steps: (1) build gene fusion graph using encompassing, or discordant, RNA-seq reads; (2) remove borders corresponding to discordant reads that show a concordant suboptimal mapping or present low weights due to multi mapping; (3) map previously unaligned RNA-seq reads between gene nodes as split-reads to obtain fusion junctions; (4) retrieve encompassing WGS reads corresponding to focal regions near fusion junctions; (5) map spanning WGS reads to focal regions using encompassing WGS reads to reconstruct genomic breakpoints. Finally, the tool produces as output the gene fusion predictions with the fusion junctions sorted according to the quantity of supporting WGS and RNA-seq reads. INTEGRATE divides fusions into tiers related to the level of sequencing support and potential biology to prioritize gene fusion candidates.[4]

4.4 Pipeline inputs

In ideal conditions, to make the pipeline usage as simple as possible, the only mandatory inputs the user must insert are the reads to be analyzed. In this case, the pipeline looks for tools' required files in default paths. If the files are found, the

gene fusion detection tools are directly executed. Otherwise, the tools are installed, and the required files are downloaded before the tools' execution. Optionally, the user can give as input the specific local paths to tools' required files.

4.4.1 Reads

FusionFlow is projected to receive as input just RNA-seq, just WGS, or both RNA-seq and WGS data. NGS technology has resulted in enormous amounts of proteomics and genomics data. These data are analyzed and reported in specific formats. Different algorithms require input data in specific formats. The pipeline supports two formats: the RNA-seq reads format must be Fastq, while the WGS data supported format is Fastq and BAM.

- **Fastq Format**: it is a text format for the storage of DNA, RNA and Genome sequence data, showing their quality. Fastq format is described in Figure 4.3. It is used in multiple and sequence alignment for Bioinformatics data collection, and it helps compare multiple sequences. In this dataset format, the first line starts with @ character that is sequence ID, the second line presents sequence letters, the third line starts with + character, the fourth line describes quality scores of the sequence letters shown in the second row. Quality score is based on ASCII (American Standard Code for Information Interchange). Fastq compressed files are in GNU zip format. Compression of quality scores is very significant for reducing storage requirements. Fastq data can be found with extensions ".fq" and ".fastq". [24]

```
@SRR001666.1 071112_SLXA-EAS1_s_7:5:1:817:345 length=36
GGGTGATGGCCGCTGCCGATGGCGTCAAATCCCACC
+SRR001666.1 071112_SLXA-EAS1_s_7:5:1:817:345 length=36
IIIIIIIIIIIIIIIIIIIIIIIIIIIII9IG9IC
```

Figure 4.3: Fastq format.

- **BAM (Binary Alignment Map) File Format**: it is used for the collection of text data into Sequence (Aligned or Unaligned) for Genome in single node elaboration for direct access. Data cannot be collected in HDFS (Hadoop

Distributed File System) without accessing BAM Library. After data is collected on Hadoop platform, Spark queries are used for sequencing data. Since BAM files are binary, they are not human readable. [24]

4.4.2 Tools' required files

Optionally, the user can give as input the tools' required files. These files must be collected in specific folders, and the user should specify the directories in the command line. If the files are found, these will be given as input directly to the tools' execution processes. Tools' required files, relative command-line arguments, and standard directories are reported in table 4.2.

Tool name	**Input files**	**Command line argument**	**Default directory**
Arriba	reference genome in Fasta format, gene annotation in GTF format, STAR index	--arriba_ref	results/arriba/files
EricScript	ensembl database of a genome	--ericscript_ref	results/ericscript/files
FusionCatcher	FusionCatcher human built data	--fusioncatcher-_ref	results/fusioncatcher/-files
GeneFuse	fusion files	--genefuse_ref	results/genefuse/files
Integrate	Integrate source code, genome index file, annotation file, built BWTs	--integrate_ref --integrate_- bwts	results/integrate/files

Table 4.2: Tools required files.

4.5 Pipeline outputs

The FusionFlow pipeline produces several files that can be divided into two categories: tools' required files and gene fusions' output files. The first category includes all the files that are needed for the execution of the tools. These files can be directly provided to the pipeline, skipping their downloads processes, or can be

downloaded running the pipeline for the first time. Then, the files will be saved in a specific path to be available to the pipeline for the subsequent runs. The second category of output includes the files that are produced as output from the gene fusion tools. Each tool gives as output one or more files in specific formats. The most diffused formats are Tab Separated Value (TSV), Variant Call Format (VCF), and standard text format.

- **TSV (Tab Separate Value)**: it is a common method to exchange data using Mail-merge functions. Every single row represents a record, and every single field value is reported as a text. Different fields in a specific record are delimited from each other by a tab character. TSV format is described in Figure 4.4. [24]

Gene1	Gene2	Breakpoint1	Breakpoint2	Site1	Site2	Sequence
GPR31	ZNF33A	6:167157143	10:38059275	CDS	3'UTR	CGCCCAGAACTCCACCAGGTGC
INSIG1	FCHSD2	7:155302678	11:73142157	exon	5'UTR	TGGCTTTGGAAAGCTTACTTAAC

Figure 4.4: TSV format.

- **VCF (Variant Call Format)**: it is designed for the collection of large Genomics data in the form of text including some special keyword, introduced with the character, in MapReduce or Spark Framework. In this format, every row is represented in the array. VCF provides a different speed when large genome data is stored depending on the Framework used. Generally, Spark allows better speed instead of MapReduce. VCF is described in Figure 4.5. [24]

Header lines
Body lines

##fileformat=VCFv4.0										
#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	SAMPLE1	SAMPLE2
2	2	.	ACG	A, AT	.	PASS	.	GT:DP	1/2:14	0/0:29
2	5	rs1	T	T, CT	.	PASS	H2; AA=T	GT:GQ	0/1:100	2/2:70

Figure 4.5: VCF format.

4.6 Pipeline configuration

4.6.1 Profiles

Nextflow allows to create one or more profiles. A profile is a set of configuration options that can be activated or chosen when launching a pipeline execution by using the -profile command line option. FusionFlow is associated with a configuration file nextflow.config that contains the description of four profiles:

- **docker**: this profile allows to run all the processes in a specific Docker container (Figure 4.6); in this case, all the virtual environments needed to install the tools and execute them are already set up in the Docker container and ready to use. The Docker image is publicly available, thus, if it is not already downloaded, it will be automatically pulled from the Docker Hub;

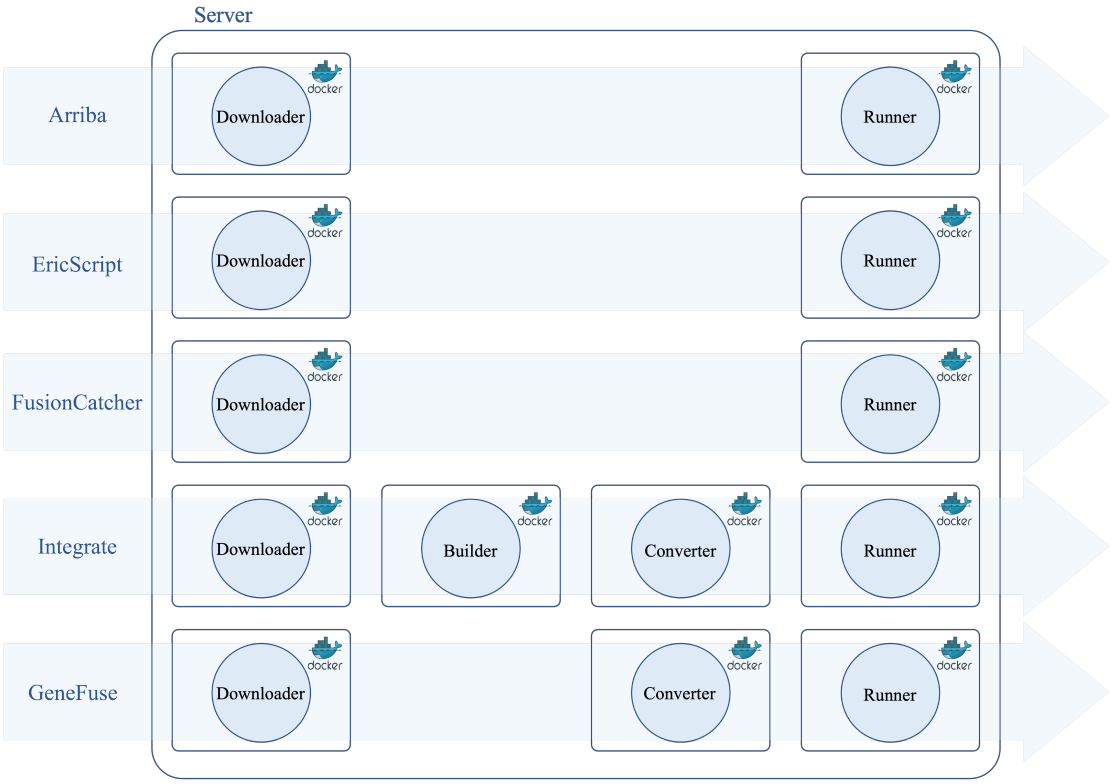

Figure 4.6: Docker profile solution.

- **local**: this profile allows to run all the processes locally. It can be used directly

inside the running Docker container or in case the virtual environments are already properly created locally (Figure 4.7, Figure 4.8);

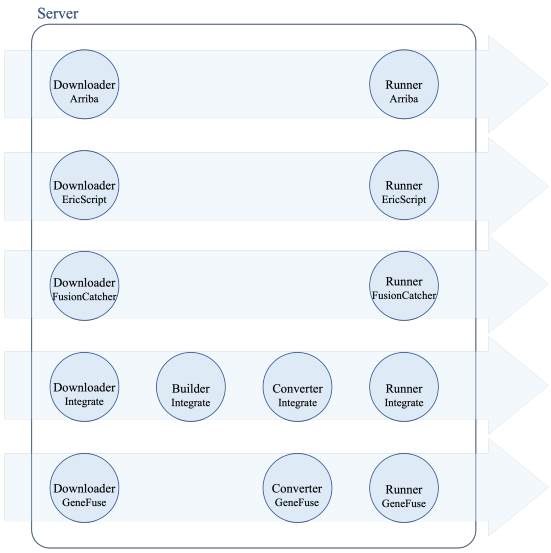

Figure 4.7: Local profile solution in server.

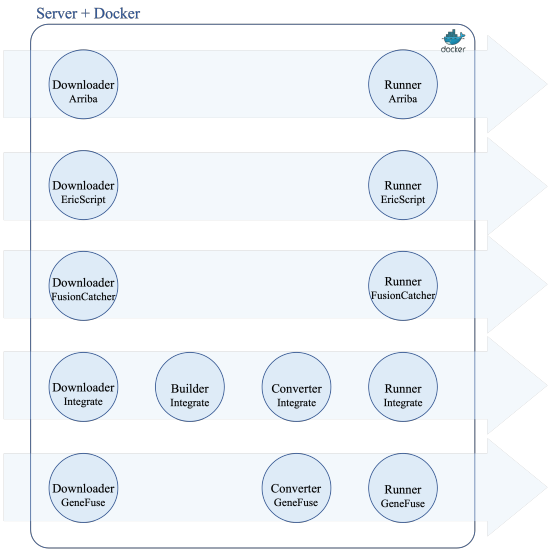

Figure 4.8: Local profile solution in Docker.

- **test_docker**: this profile allows to test the docker profile using test samples;
- **test_local**: this profile allows to test the local profile using test samples.

4.6.2 Other attributes

The nextflow.config configuration file sets all the other standard arguments used in the different processes. However, these parameters are available at the command line. The command-line arguments, a brief description, and the default value are shown in Table 4.3.

Argument name	Description	Default value
--help	Flag to show the help message.	false

--dnabam	Flag to specify the dna reads are in bam format.	false
--single_end	Flag to specify reads are single-end.	false
--outdir	The utput directory where the results will be saved.	"results/"
--referenceGenome	The path to the reference genome	"results/reference_genome/-hg38.fa"
--referenceGenome_index	The path to the reference genome indexes.	"results/reference_genome/index"
--ericscript_ref	The path to the EricScript files.	"results/ericscript/files/ericscript-_db_homosapiens_ensembl84"
--arriba_ref	The path to Arriba files.	"results/arriba/files"
--fusioncatcher_ref	The path to FusionCatcher files.	"results/fusioncatcher/files"
--integrate_ref	The path to Integrate files.	"results/integrate/files"
--integrate_bwts	The path to Integrate bwts files.	"results/integrate/files/bwts"
--genefuse_ref	The path to GeneFuse files.	"results/genefuse/files"
--rnareads	The path to RNA-seq reads.	""
--dnareads_tumor	The path to WGS tumour reads.	""
--dnareads_normal	The path to WGS normal reads.	""
--envPath_ericscript	The path to EricScript virtual environment..	"/opt/conda/envs/ericscript/bin"
--envPath_arriba	The path to Arriba virtual environment.	"/opt/conda/envs/arriba/"
--envPath_fusioncatcher	The path to FusionCatcher virtual environment.	"/opt/conda/envs/fusioncatcher/bin"

--envPath_integrate	The path to Integrate virtual environment.	"/opt/conda/envs/integrate/bin"
--envPath_genefuse	The path to GeneFuse virtual environment.	"/opt/conda/envs/genefuse/bin"
--nthreads	Number of threads.	8
--arriba	Flag to run Arriba.	false
--ericscript	Flag to run EricScript.	false
--fusioncatcher	Flag to run FusionCatcher.	false
--genefuse	Flag to run GeneFuse.	false
--integrate	Flag to run INTEGRATE.	false

Table 4.3: Pipeline attributes.

4.7 Pipeline execution

The pipeline can be executed by entering in the shell terminal the command shown in Figure 4.9.

```
nextflow run federicacitarrella/FusionFlow \
      --rnareads “/path/to/rna/reads_{1,2}.*” \
      --dnareads_tumor “/path/to/dna/tumor/reads_{3,4}.*” \
      --dnareads_normal “/path/to/dna/normal/reads_{5,6}.*” \
      --arriba --ericscript --fusioncatcher --integrate --genefuse \
      -profile <docker/local/test_docker/test_local>
```

Figure 4.9: Shell standard execution command.

Firstly, the path to Nextflow executable file is specified, followed by the run command and pipeline path. Then paths to RNA and DNA reads must be included. The tools flags should be specified just if not all the tools are involved in the current analysis. Otherwise, all the tools will be involved in the gene fusions detection if no flag is specified. Then other optional arguments can be included, and lastly, the profile definition.
Launching the described command, Nextflow looks for the main pipeline file in the path specified. If that file does not exist, Nextflow searches for a public repository

with the same name on GitHub. If it is found, the repository is automatically downloaded and the pipeline executed. The repository found is stored in the Nextflow home directory, which is by default the $HOME/.nextflow path, and thus will be called for any further executions.

4.7.1 Files preparation

The FusionFlow pipeline can be run for three different purposes depending on the type of input received:

1. detect gene fusions in RNA data;
2. detect gene fusions in DNA data;
3. detect gene fusions in RNA and DNA data.

In the first case, just RNA data are needed to execute the pipeline. The input must be paired-end Fastq files. The two RNA mates files must be named with the pair ID, followed by "_1" and "_2" suffix, before the extension characters (e.g., SRR12345_1.fq.gz and SRR12345_2.fq.gz).
In the second case, just DNA data are required. The DNA data can be given in input as paired-end Fastq files or as BAM files. If they are given as Fastq files, the two DNA tumor mates files must be named with the pair ID, followed by "_3" and "_4" suffix, while the two DNA normal mates files must be named with the pair ID, followed by "_5" and "_6" suffix.
In the third case, both RNA and DNA data are needed. They must be named as described earlier for the other cases.
Quotes are always required to enclose files paths in the command line.

4.7.2 Running modes

The pipeline can be run in two modes:

- single mode;
- batch mode.

In single mode, the pipeline receives as input and analyzes just a file pair. The batch mode is used when the pipeline receives input files with more than one pair ID. In this case, the pipeline sequentially executes the analysis for every pair reads. For each pair ID, the outputs are stored in the output path into specific folders named as the pair ID associated to distinguish the various output.

4.8 Minimum system requirements

FusionFlow includes five gene fusion detection tools. For each tool, the pipeline can perform both the installation and the tool execution. These steps require different minimum specifications.
Minimum system requirements of three different conditions are reported:

- To execute the entire pipeline >=8 CPU cores and >=45GB RAM are required.
- To run at least the installation and the execution of a single tool (EricScript) >=4 CPU cores and >=16GB RAM are required.
- To run just the tools execution, skipping the installations >=8 CPU cores and >=31GB RAM are required.

4.9 Pipeline availability and GitHub support

The FusionFlow pipeline for gene fusion detection from RNA-seq and WGS data has been developed using Nextflow and Docker, and can be executed on any UNIX-based OS with Nextflow and Docker installed. The workflow is freely available on GitHub (https://github.com/federicacitarrella/FusionFlow.git), and the Docker image required for execution can be found on the DockerHub server (https://hub.docker.com/repository/docker/federicacitarrella/pipeline).

Nextflows integrates with GitHub hosted code repositories. This feature allows to manage the project code more consistently and to use it quickly and transparently. The organization of the files within the GitHub repository is shown in Figure 4.10. In the /federicacitarrella/FusionFlow path, there are the actual Nextflow pipeline file pipeline.nf and the configuration file nextflow.config. In the same directory,

there are also the Dockerfile, a shell script to build the Docker image build.sh, a text file README.txt and finally the five YML files, one for each gene fusion detection tool, to create the Conda virtual environments. A folder named "tutorial" contains all the test files used in test_docker and test_local profiles to verify the correct functioning of the pipeline.

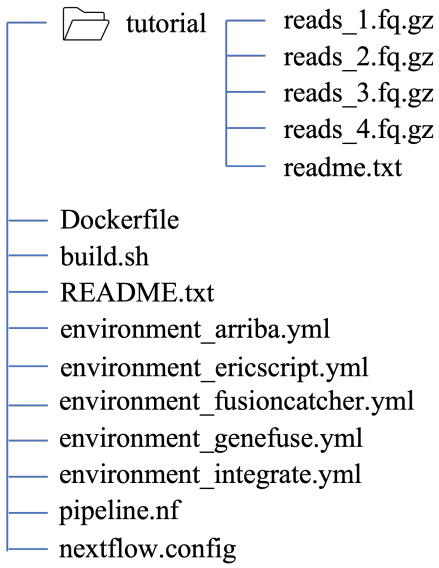

Figure 4.10: GitHub files' organization.

Chapter 5

Pipeline architecture

5.1 General architecture

The pipeline run is based on two files, the Nextflow pipeline file *pipeline.nf* and the configuration file *nextflow.config*. The Nextflow *pipeline.nf* file contains the actual pipeline, while the *nextflow.config* file provides the configuration needed to run the pipeline. The pipeline is composed of fifteen processes. These processes can be divided into three main categories:

- **downloaders**: are responsible for the tools installation and download input files. The downloaders processes are: *referenceGenome_downloader*, *arriba_downloader*, *ericscript_downloader*, *fusioncatcher_downloader*, *integrate_downloader* and *genefuse_downloader*;
- **converters**: are responsible for the file preparation and format conversion if needed. The converters processes are: *integrate_converter* and *genefuse_converter*;
- **runners**: allow the code and tools execution. The runner processes are: *arriba*, *ericscript*, *fusioncatcher*, *integrate*, *genefuse*, *referenceGenom_index*, *integrate_builder*.

The fifteen processes are structured into six main parallel lines shown in green in Figure 5.1.
Executing the script with Nextflow, the algorithm will look for the required files in the paths specified in *nextflow.config* configuration file or in the paths specified in

the command line. If the files are found, the associated downloader is skipped, and the next processes are run.

Nextflow processes are normally executed concurrently. Nextflow queue channels are used to execute sequentially downloaders, converters, and runners and provide inter-communication between processes. A queue channel creates an asynchronous unidirectional FIFO queue and allows to connect two processes or operators. The usage of a combination of queue channels permits the creation of predefined sequences of processes. The processes expect to receive inputs data from the channels specified in the input block. When the inputs are emitted, the processes run.

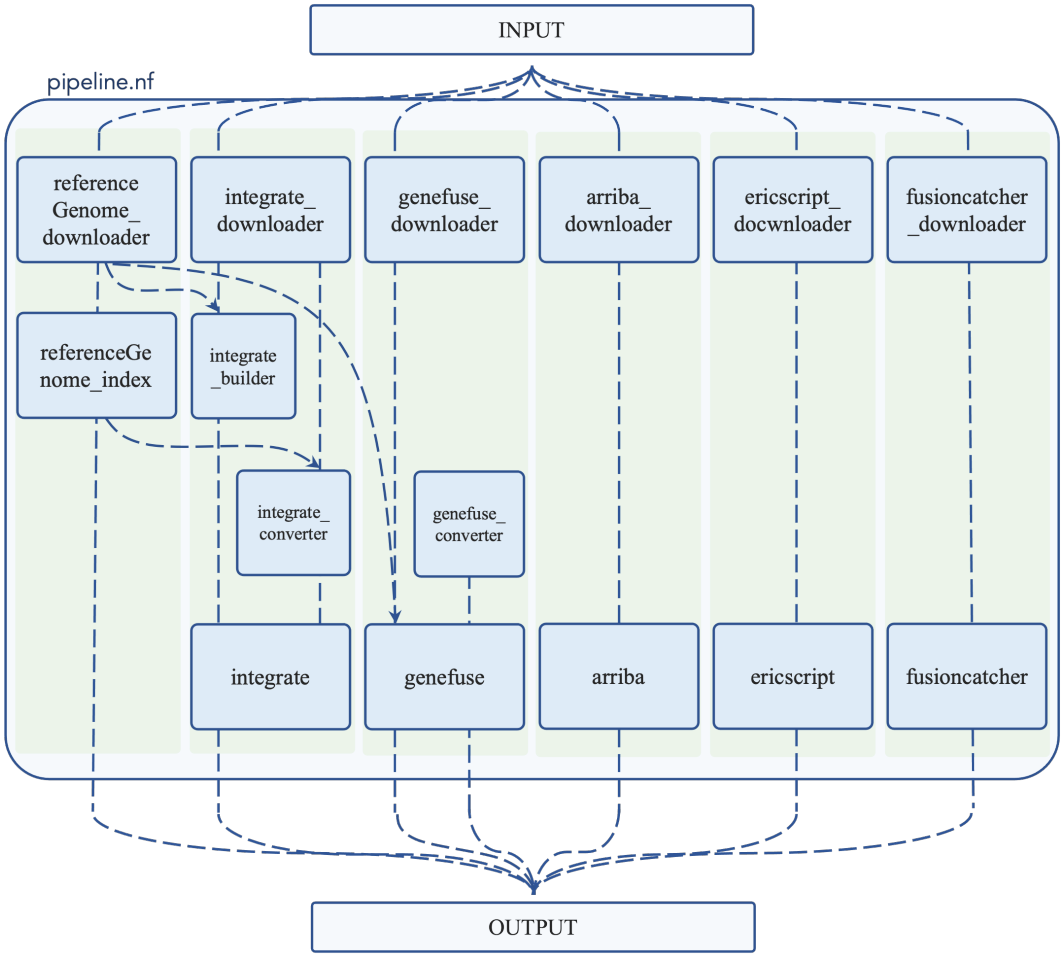

Figure 5.1: Pipeline architecture parallelization.

5.2 Tools architecture

FusionFlow includes five fusion detection tools. Each tool in the pipeline has associated several processes that communicate through Nextflow queue channels. The processes are triggered by input and, after the script block execution, provide output to trigger the subsequent processes.

5.2.1 EricScript

EricScript is a bioinformatic framework for the discovery of gene fusions in paired end RNA-seq data.

Two pipeline processes are associated with EricScript: *ericscript_downloader*

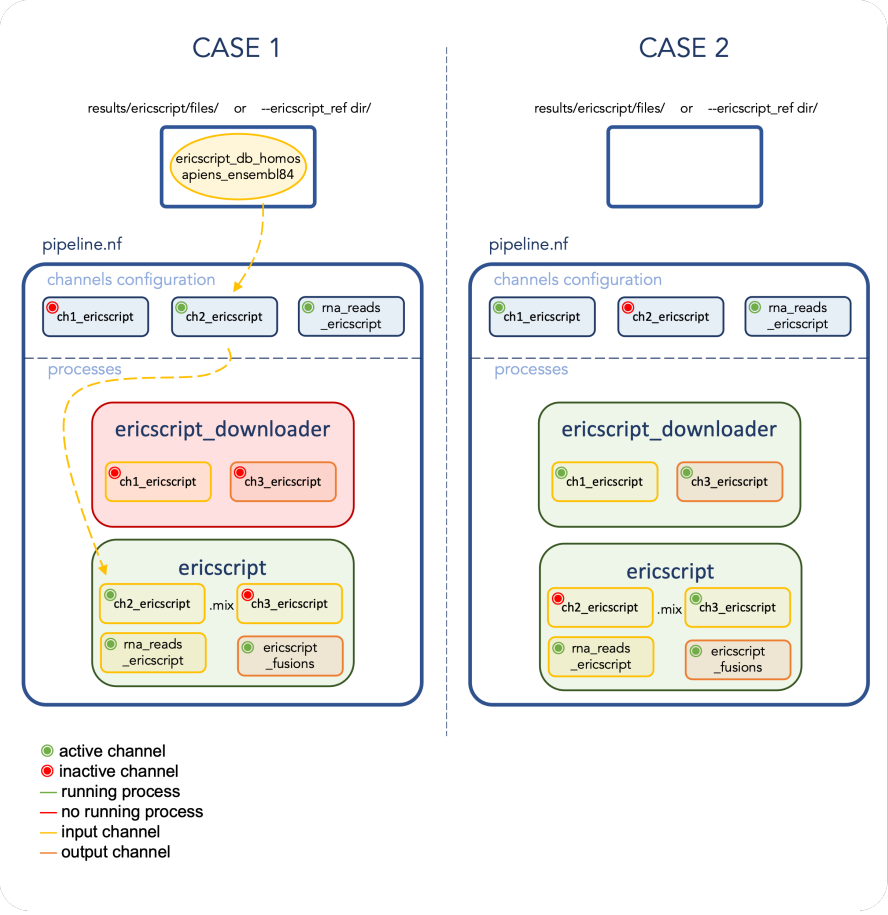

Figure 5.2: EricScript processes architecture.

and *ericscript*. Six channels are associated with EricScript: *rna_reads_ericscript*, *input_ch_ericscript*, *ch1_ericscript*, *ch2_ericscript*, *ch3_ericscript* and *ericscript-_fusions*.

Firstly, the *rna_reads_ericscript* channel is created. If RNA data are given as input through the rnareads Nextflow parameter, the *rna_reads_ericscript* channel is created using the fromFilePairs function. While, if no RNA data are given as input, the *rna_reads_ericscript* channel is created as an empty channel blocking the ericscript process execution. Then the *input_ch_channel* is created. The path specified in the *ericscript_ref* param is used to look for the EricScript database. Depending on the existence of the ericscript database, different cases are possible as shown in Figure 5.2. If the defined database is found, then the database is given as input to the *input_ch_ericscript* channel, the *ch1_ericscript* is created as an empty channel and the *ch2_ericscript* channel is created equal to the *input_ch_channel*. Being empty, the *ch1_ericscript* does not trigger the *ericscript_downloader*, while the *ch2_ericscript* is activated and triggers the ericscript runner process. If the database is not found, then the *ch1_ericscript* channel is activated and the *ch2_ericscript* channel is set to empty. Thus, the *ch1_ericscript* channel triggers the downloader that downloads the database and decompresses it and gives it as input to *ch3_ericscript* that emits the database to the ericscript runner. Then, after the ericscript script block execution, the output produced is given as input to the *ericscript_fusions* channel and it is stored in the *outdir* local directory (Listing 5.1).

```
1  refFile_ericscript = file(params.ericscript_ref)
2  params.skip_ericscript = refFile_ericscript.exists()
3
4  rna_reads_ericscript = ( params.rnareads ? [Channel.fromFilePairs(
       params.rnareads)] : [Channel.empty()])
5  Channel.fromPath(params.ericscript_ref).set{ input_ch_ericscript }
6  (ch1_ericscript, ch2_ericscript) = ( params.skip_ericscript ? [
       Channel.empty(), input_ch_ericscript] : [input_ch_ericscript,
       Channel.empty()] )
7
8  process ericsctipt_downloader{
9      tag "Downloading"
10
11     publishDir "${params.outdir}/ericscript/references", mode: '
       copy'
12
```

```
13      input:
14      val x from ch1_ericscript
15
16      output:
17      file "ericscript_db_homosapiens_ensembl84" into ch3_ericscript
18
19      when: params.ericscript || params.all
20
21      script:
22      """
23      #!/bin/bash
24      export PATH='${params.envPath_ericscript}:$PATH'
25
26      gdown 'https://drive.google.com/uc?export=download&confirm=
        qgOc&id=1VENACpUv_81HbIB8xZN0frasrAS7M4SP'
27      tar -xf ericscript_db_homosapiens_ensembl84.tar.bz2
28      rm ericscript_db_homosapiens_ensembl84.tar.bz2
29      """
30  }
31  process ericscript{
32      tag "${pair_id}"
33
34      publishDir "${params.outdir}/ericscript", mode: 'copy'
35
36      input:
37      tuple pair_id, file(rna_reads), file(ericscript_db) from
        rna_reads_ericscript.combine(ch2_ericscript.mix(ch3_ericscript)
        )
38
39      output:
40      file "output/${pair_id}" optional true into ericscript_fusions
41
42      when: params.ericscript || params.all
43
44      script:
45      reads = params.single_end ? rna_reads[0] : "../${rna_reads[0]}
        ../${rna_reads[1]}"
46      """
47      #!/bin/bash
48      export PATH='${params.envPath_ericscript}:$PATH'
49
50      mkdir output && cd output
51      ericscript.pl -o ./${pair_id} -db ../${ericscript_db} ${reads}
52      """
53  }
```

Listing 5.1: EricScript processes.

5.2.2 Arriba

Arriba is a command-line tool for the detection of gene fusions from RNA-Seq data. Two pipeline processes are associated with Arriba: *arriba_downloader* and *arriba*. Six channels are associated with Arriba: *rna_reads_arriba*, *input_ch_arriba*, *ch1_arriba*, *ch2_arriba*, *ch3_arriba* and *arriba_fusions*.

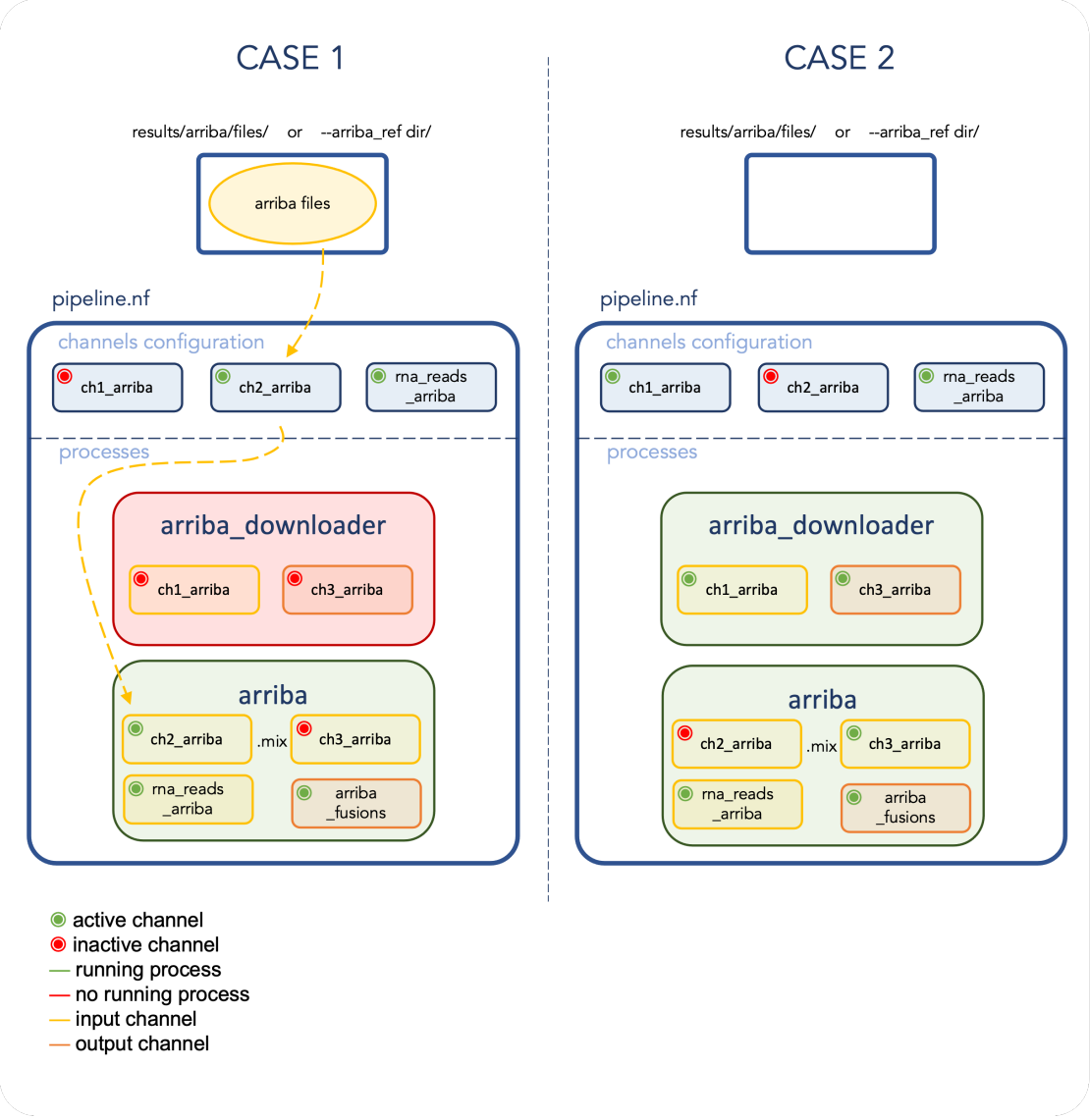

Figure 5.3: Arriba processes architecture.

The Arriba architecture is similar to the Ericscript structure. Firstly, the *rna_reads_arriba* channel is created. If RNA data are given as input through the rnareads Nextflow parameter, the *rna_reads_arriba* channel is created using the fromFilePairs function. While, if no RNA data are given as input, the *rna_reads_arriba* channel is created as an empty channel blocking the arriba process execution. Then the *input_ch_arriba* is created. The path specified in the *arriba_ref* param is used to look for the Arriba required files. Depending on the existence of the files directory, different cases are possible as shown in Figure 5.3. If the defined directory is found, then the files are given as input to the *input_ch_arriba* channel, the *ch1_arriba* is created as an empty channel and the *ch2_arriba* channel is created equal to the *input_ch_arriba*. Being empty, the *ch1_arriba* does not trigger the *arriba_downloader*, while the *ch2_arriba* is activated and triggers the arriba runner process. If the directory is not found, then the *ch1_arriba* channel is activated and the *ch2_arriba* channel is set to empty. Thus, the *ch1_arriba* channel triggers the downloader that downloads the required files and gives them as input to *ch3_arriba* which emits the database to the arriba runner. Then, after the arriba script block execution, the output produced is given as input to the *arriba_fusions channel* (Listing 5.2).

```
refDir_arriba = file(params.arriba_ref)
params.skip_arriba = refDir_arriba.exists()

rna_reads_ericscript = ( params.rnareads ? [Channel.fromFilePairs(
    params.rnareads)] : [Channel.empty()])
Channel.fromPath(params.arriba_ref).set{ input_ch_arriba }
(ch1_arriba, ch2_arriba) = ( params.skip_arriba ? [Channel.empty()
    , input_ch_arriba] : [input_ch_arriba, Channel.empty()] )

process arriba_downloader{
    tag "Downloading"

    publishDir "${params.outdir}/arriba", mode: 'copy'

    input:
    val x from ch1_arriba

    output:
    file "references" into ch3_arriba

    when: params.arriba || params.all
```

```
20
21      shell:
22      '''
23      #!/bin/bash
24      export PATH="!{params.envPath_arriba}bin:$PATH"
25
26      mkdir references && cd "$_"
27      !{params.envPath_arriba}var/lib/arriba/download_references.sh
        GRCh38+ENSEMBL93
28      '''
29  }
30  process arriba{
31      tag "${pair_id}"
32
33      publishDir "${params.outdir}/arriba", mode: 'copy'
34
35      input:
36      tuple pair_id, file(rna_reads), file(arriba_ref) from
        rna_reads_arriba.combine(ch2_arriba.mix(ch3_arriba))
37
38      output:
39      file "output/${pair_id}" optional true into arriba_fusions
40
41      when: params.arriba || params.all
42
43      script:
44      """
45      #!/bin/bash
46      export PATH='${params.envPath_arriba}bin:$PATH'
47
48      run_arriba.sh ${arriba_ref}/STAR_index_GRCh38_ENSEMBL93/ ${
        arriba_ref}/ENSEMBL93.gtf ${arriba_ref}/GRCh38.fa ${params.
        envPath_arriba}var/lib/arriba/blacklist_hg19_hs37d5_GRCh37_v2
        .1.0.tsv.gz ${params.envPath_arriba}var/lib/arriba/
        known_fusions_hg19_hs37d5_GRCh37_v2.1.0.tsv.gz ${params.
        envPath_arriba}var/lib/arriba/
        protein_domains_hg19_hs37d5_GRCh37_v2.1.0.gff3 ${params.
        nthreads} ${rna_reads}
49      mkdir output && mkdir output/${pair_id}
50      mv *.out output/${pair_id}
51      mv *.tsv output/${pair_id}
52      mv *.out output/${pair_id}
53      mv *bam* output/${pair_id}
54      """
55  }
```

Listing 5.2: Arriba processes.

5.2.3 FusionCatcher

FusionCatcher is a bioinformatcs tool that searches for novel or known somatic fusion genes, translocations, and chimeras in RNA-seq data from diseased samples. Two pipeline processes are associated with FusionCatcher: *fusioncatcher_downloader* and *fusioncatcher*. Six channels are associated with FusionCatcher: *rna_reads_fusioncatcher*, *input_ch_fusioncatcher*, *ch1_fusioncatcher*, *ch2_fusioncatcher*, *ch3_fusioncatcher* and *fusioncatcher_fusions*.

The FusionCatcher architecture is similar to the other RNA tools. Firstly, the

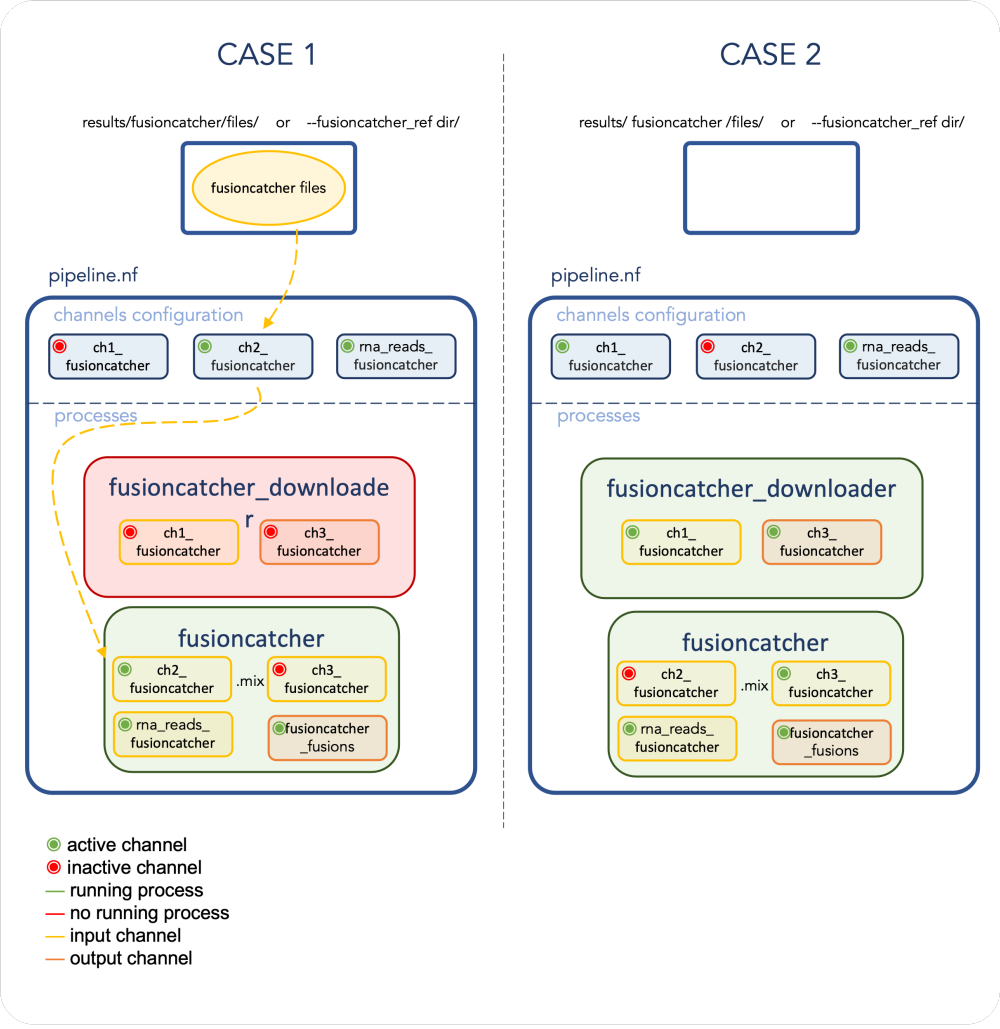

Figure 5.4: FusionCatcher processes architecture.

rna_reads_fusioncatcher channel is created. If RNA data are given as input through the rnareads Nextflow parameter, the *rna_reads_fusioncatcher* channel is created using the fromFilePairs function. While, if no RNA data are given as input, the *rna_reads_fusioncatcher* channel is created as an empty channel blocking the fusioncatcher process execution. Then the *input_ch_fusioncatcher* is created. The path specified in the *fusioncatcher_ref* param is used to look for the FusionCatcher required files. Depending on the existence of the files directory, different cases are possible as shown in Figure 5.4. If the defined directory is found, then the files are given as input to the *input_ch_fusioncatcher* channel, the *ch1_fusioncatcher* is created as an empty channel and the *ch2_fusioncatcher* channel is created equal to the *input_ch_fusioncatcher*. Being empty, the *ch1_fusioncatcher* does not trigger the *fusioncatcher_downloader*, while the *ch2_fusioncatcher* is activated and triggers the fusioncatcher runner process. If the directory is not found, then the *ch1_fusioncatcher* channel is activated and the *ch2_fusioncatcher* channel is set to empty. Thus, the *ch1_fusioncatcher* channel triggers the downloader that downloads the required files and gives them as input to *ch3_fusioncatcher* which emits the database to the fusioncatcher runner. Then, after the fusioncatcher script block execution, the output produced is given as input to the *fusioncatcher_fusions* channel (Listing 5.3).

```
1  refDir_fusioncatcher = file(params.fusioncatcher_ref)
2  params.skip_fusioncatcher= refDir_fusioncatcher.exists()
3  rna_reads_fusioncatcher = ( params.rnareads ? [Channel.
       fromFilePairs(params.rnareads)] : [Channel.empty()])
4  Channel.fromPath(params.fusioncatcher_ref).set{
       input_ch_fusioncatcher }
5  (ch1_fusioncatcher , ch2_fusioncatcher) = ( params.
       skip_fusioncatcher ? [Channel.empty(), input_ch_fusioncatcher]
       : [input_ch_fusioncatcher, Channel.empty()] )
6
7  process fusioncatcher_downloader{
8      tag "Downloading"
9
10     publishDir "${params.outdir}/fusioncatcher", mode: 'copy'
11
12     input:
13     val x from ch1_fusioncatcher
14
15     output:
```

```
16      file "references" into ch3_fusioncatcher
17
18      when: params.fusioncatcher || params.all
19
20      shell:
21      '''
22      #!/bin/bash
23      mkdir -p references && cd "$_"
24      wget http://sourceforge.net/projects/fusioncatcher/files/data/
        human_v102.tar.gz.aa
25      wget http://sourceforge.net/projects/fusioncatcher/files/data/
        human_v102.tar.gz.ab
26      wget http://sourceforge.net/projects/fusioncatcher/files/data/
        human_v102.tar.gz.ac
27      wget http://sourceforge.net/projects/fusioncatcher/files/data/
        human_v102.tar.gz.ad
28      cat human_v102.tar.gz.* | tar xz
29      ln -s human_v102 current
30      '''
31  }
32  process fusioncatcher{
33      tag "${pair_id}"
34
35      publishDir "${params.outdir}/fusioncatcher", mode: 'copy'
36
37      input:
38      tuple pair_id, file(rna_reads), file(fusioncatcher_db) from
        rna_reads_fusioncatcher.combine(ch2_fusioncatcher.mix(
        ch3_fusioncatcher))
39
40      output:
41      file "output/${pair_id}" optional true into
        fusioncatcher_fusions
42
43      when: params.fusioncatcher || params.all
44
45      script:
46      reads = params.single_end ? rna_reads[0] : "${rna_reads[0]},${
        rna_reads[1]}"
47      """
48      #!/bin/bash
49      export PATH='${params.envPath_fusioncatcher}:$PATH'
50
51      fusioncatcher -d ${fusioncatcher_db}/human_v102 -i ${reads} -o
        output/${pair_id}
52      """
53  }
```

Listing 5.3: FusionCatcher processes.

5.2.4 GeneFuse

GeneFuse is a tool to detect target gene fusions by scanning Fastq DNA files.
In FusionFlow three processes are associated with GeneFuse: *genefuse_downloader*,

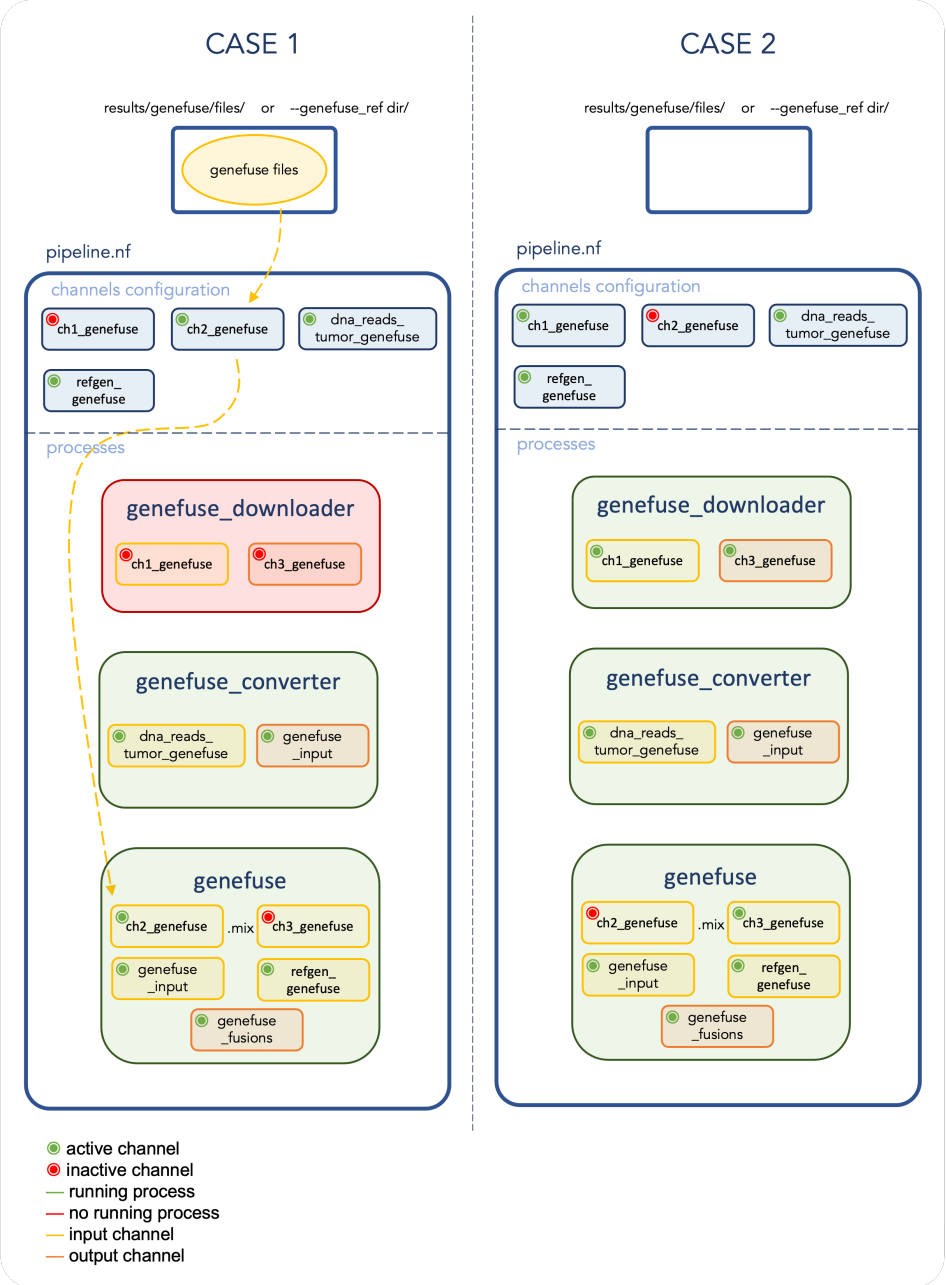

Figure 5.5: GeneFuse processes architecture.

genefuse_converter and *genefuse*. *genefuse_converter* process is also dependent on *referenceGenome_downloader*. Thirteen channels are associated with GeneFuse: *dna_reads_tumor_genefuse*, *input_ch1_genefuse*, *input_ch2_genefuse*, *refgen_genefuse*, *refgen_genefuse_down*, *ch1_genefuse*, *ch2_genefuse*, *ch3_genefuse*, *input_ch1_genefuse*, *input_ch2_genefuse*, *genefuse_input* and *genefuse_fusions*. Firstly, the *dna_reads_tumor_genefuse* channel is created. If DNA data are given as input through the *dnareads_tumor* Nextflow parameter, the *dna_reads_tumor_genefuse* channel is created using the fromFilePairs function. While, if no DNA data are given as input, the *dna_reads_tumor_genefuse* channel is created as an empty channel blocking the genefuse process execution. Subsequently, the *input_ch1_genefuse*, *input_ch2_genefuse*, *refgen_genefuse*, *ch1_genefuse* and *ch2_genefuse* channels are created. The paths specified in the *genefuse_ref* and referenceGenome params are used to look for the GeneFuse required files. Depending on the existence of the files directories, different cases are possible as shown in Figure 5.5. If the defined directories are found, then the files are given as input to the previously mentioned channels, triggering the running processes. If the directories are not found, then the downloader processes are run. In this case, the runners are executed after the downloader execution. After the genefuse script block execution, the output produced is given as input to the *genefuse_fusions* channel (Listing 5.4).

```
1  refDir_refgen = file(params.referenceGenome)
2  refDir_genefuse = file(params.genefuse_ref)
3  params.skip_refgen = refDir_refgen.exists()
4  params.skip_genefuse = refDir_genefuse.exists()
5  Channel.fromPath(params.referenceGenome).into{ input_ch4_refgen }
6  Channel.fromPath(params.genefuse_ref).set{ input_ch1_genefuse }
7  dna_reads_tumor_genefuse = ( params.dnareads_tumor ? [Channel.
       fromFilePairs(params.dnareads_tumor)] : [Channel.empty()] )
8  (refgen_downloader, refgen_genefuse) = ( params.skip_refgen ? [
       Channel.empty(), input_ch4_refgen] : [input_ch4_refgen, Channel
       .empty()] )
9  (ch1_genefuse , ch2_genefuse) = ( params.skip_genefuse ? [Channel.
       empty(), input_ch1_genefuse] : [input_ch1_genefuse, Channel.
       empty()] )
10
11 process genefuse_downloader{
12     tag "Downloading"
13
14     publishDir "${params.outdir}/genefuse", mode: 'copy'
```

```
15
16      input:
17      val x from ch1_genefuse
18
19      output:
20      file "references" into ch3_genefuse
21
22      when: params.genefuse || params.all
23
24      shell:
25      '''
26      #!/bin/bash
27      export PATH="!{params.envPath_integrate}:$PATH"
28
29      mkdir references && cd "$_"
30      gdown "https://drive.google.com/uc?export=download&confirm=
        qgOc&id=1OBLTo-yGZ88UGcF0F3v_7n8mLTQblWg8"
31      chmod a+x ./genefuse
32      gdown "https://drive.google.com/uc?export=download&confirm=
        qgOc&id=1eRI5lAw0qntj0EbEpaNpvRA7saw_iyY3"
33      '''
34  }
35  process genefuse_converter{
36      tag "${pair_id}"
37
38      publishDir "${params.outdir}/genefuse", mode: 'copy'
39
40      input:
41      tuple pair_id, file(wgstinput) from dna_reads_tumor_genefuse
42
43      output:
44      tuple pair_id, file("input/${pair_id}") into genefuse_input
45
46      when: params.genefuse || params.all
47
48      script:
49      """
50      #!/bin/bash
51      export PATH='${params.envPath_genefuse}:$PATH'
52
53      mkdir input && mkdir input/${pair_id}
54      if ${params.dnabam} && ${integrateWGSt}; then
55          samtools sort -n  -o ${wgstinput}
56          samtools fastq -@ ${params.nthreads} ${wgstinput} -1 ${
        pair_id}_3.fq.gz -2 ${pair_id}_4.fq.gz -0 /dev/null -s /dev/
        null -n
57          cp *.fq.gz input/${pair_id}
58      elif ${integrateWGSt}; then
59          cp ${wgstinput} input/${pair_id}
```

```
60      fi
61      """
62  }
63  process genefuse{
64      tag "${pair_id}"
65
66      publishDir "${params.outdir}/genefuse", mode: 'copy'
67
68      input:
69      tuple pair_id, file(input), file(refgen), file(genefuse_db)
        from genefuse_input.combine(refgen_genefuse.mix(
        refgen_genefuse_down)).combine(ch2_genefuse.mix(ch3_genefuse))
70
71      output:
72      file "output/${pair_id}" optional true into genefuse_fusions
73
74      when: params.genefuse || params.all
75
76      script:
77      """
78      #!/bin/bash
79      export PATH='${params.envPath_genefuse}:$PATH'
80
81      cp ${input}/* .
82      ${genefuse_db}/genefuse -r ${refgen} -f ${genefuse_db}/
        druggable.hg38.csv -1 ${pair_id}_3.* -2 ${pair_id}_4.* -h
        report.html > result
83      mkdir output && mkdir output/${pair_id}
84      cp report.html output/${pair_id}
85      cp result output/${pair_id}
86      """
87  }
```

Listing 5.4: GeneFuse processes.

5.2.5 INTEGRATE

INTEGRATE is a tool for gene fusions discovery that finds exact fusion junctions and genomic breakpoints by combining RNA-Seq and WGS data.

In FusionFlow four processes are associated with INTEGRATE: *integrate_downloader*, *integrate_builder*, *integrate_converter* and *integrate*. *integrate_builder* and *integrate_converter* processes are also dependent on *referenceGenome_downloader* and *referenceGenome_index* processes. Twenty-three channels are associated with INTEGRATE: *rna_reads_integrater*, *dna_reads_tumor_integrate*, *dna_reads_normal_integrate*, *input_ch1_integrate*, *input_ch2_integrate*, *input_ch3_integrate*,

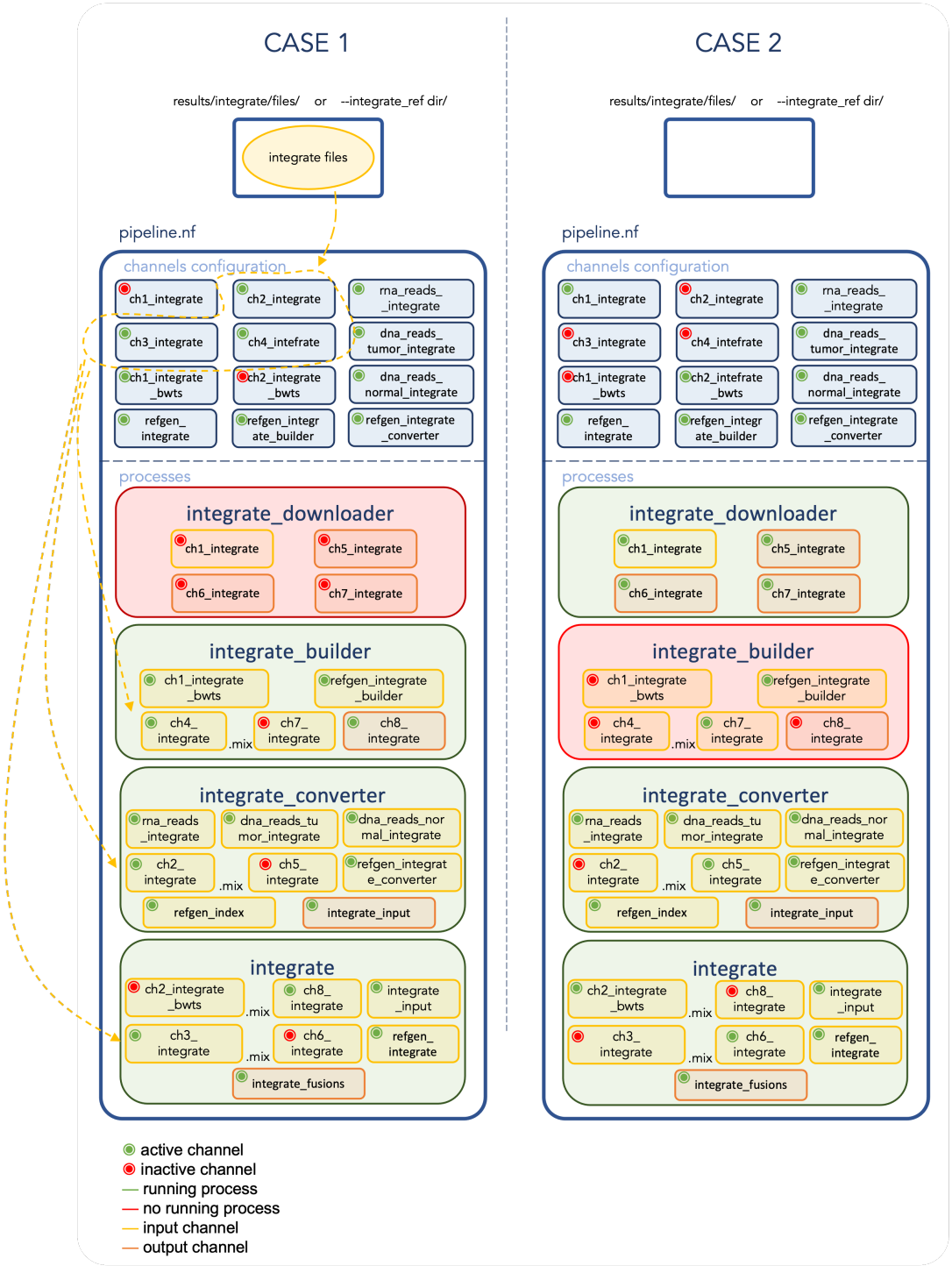

Figure 5.6: Integrate processes architecture.

input_ch1_bwts, *refgen_integrate*, *refgen_integrate_builder*, *refgen_integrate_builder_down*, *refgen_integrate_converter*, *ch1_integrate*, *ch2_integrate*, *ch3_integrate*, *ch4_integrate*, *ch5_integrate*, *ch6_integrate*, *ch7_integrate*, *ch8_integrate*, *refgen_index*, *refgen_index_down*, *integrate_input* and *integrate_fusions*.
Firstly, the *rna_reads_integrate channel* is created. If RNA data are given as input through the rnareads Nextflow parameter, the *rna_reads_integrate* channel is created using the fromFilePairs function. While, if no RNA data are given as input, the *rna_reads_integrate* channel is created as an empty channel blocking the integrate process execution. Then the same process is repeated for *dna_reads_tumor_integrate* and *dna_reads_normal_integrate* channels. Subsequently, the *input_ch1_integrate*, *input_ch2_integrate*, *input_ch3_integrate*, *input_ch1_bwts*, *refgen_integrate*, *refgen_integrate_builder*, *refgen_integrate- _converter* channels are created. The paths specified in the *integrate_ref*, *integrate_bwts*, referenceGenome params are used to look for the Integrate required files. Depending on the existence of the files directories, different cases are possible as shown in Figure 5.6. If the defined directories are found, then the files are given as input to the previously mentioned channels, triggering the running processes. If the directories are not found, then the downloader processes are run. In this case, the runners are executed after the downloader execution. After the integrate script block execution, the output produced is given in input to the *integrate_fusions* channel (Listing 5.5).

```
refDir_refgen = file(params.referenceGenome)
refDir_refgen_index = file(params.referenceGenome_index)
refDir_integrate = file(params.integrate_ref)
refDir_integrate_bwts = file(params.integrate_bwts)
params.skip_refgen = refDir_refgen.exists()
params.skip_refgen_index = refDir_refgen_index.exists()
params.skip_integrate = refDir_integrate.exists()
params.skip_integrate_bulder = refDir_integrate_bwts.exists()

integrateWGSt = false
integrateWGSn = false
command1 = ""
command2 = ""
if (params.dnareads_tumor) {
  integrateWGSt = true
  command1 = "dna.tumor.bam"
}
if (params.dnareads_normal) {
```

```
  integrateWGSn = true
  command2 = "dna.normal.bam"
}

rna_reads_integrate = ( params.rnareads ? [Channel.fromFilePairs(
    params.rnareads),] : [Channel.empty()])
dna_reads_tumor_integrate = ( params.dnareads_tumor ? [Channel.
    fromFilePairs(params.dnareads_tumor)] : [support1.map{id,reads
    -> tuple(id,"1")}] )
dna_reads_normal_integrate = ( params.dnareads_normal ? [Channel.
    fromFilePairs(params.dnareads_normal)] : [support2.map{id,reads
     -> tuple(id,"1")}] )
Channel.fromPath(params.referenceGenome).into{ input_ch1_refgen ;
    input_ch2_refgen ; input_ch3_refgen ; input_ch4_refgen ;
    input_ch5_refgen}
Channel.fromPath(params.referenceGenome_index).set{
    input_ch1_refgen_index }
Channel.fromPath(params.integrate_ref).into{ input_ch1_integrate;
    input_ch2_integrate;input_ch3_integrate }
Channel.fromPath(params.integrate_bwts).set{ input_ch1_bwts }
(refgen_integrate , refgen_integrate_builder,
    refgen_integrate_converter) = ( params.skip_refgen ? [
    input_ch1_refgen, input_ch2_refgen, input_ch3_refgen] : [
    Channel.empty(), Channel.empty(), Channel.empty()] )
(refgen_index_trigger , refgen_index) = ( (params.
    skip_refgen_index || params.dnabam) ? [Channel.empty(),
    input_ch1_refgen_index] : [input_ch1_refgen_index, Channel.
    empty()] )
(ch1_integrate , ch2_integrate , ch3_integrate, ch4_integrate) = (
     params.skip_integrate ? [Channel.empty(), input_ch1_integrate,
     input_ch2_integrate, input_ch3_integrate] : [
    input_ch1_integrate, Channel.empty(), Channel.empty(), Channel.
    empty()] )
(ch1_integrate_bwts , ch2_integrate_bwts) = ( params.
    skip_integrate_bulder ? [Channel.empty(), input_ch1_bwts] : [
    input_ch1_bwts, Channel.empty()] )

process integrate_downloader{
    tag "Downloading"

    publishDir "${params.outdir}/integrate", mode: 'copy'

    input:
    val x from ch1_integrate

    output:
    file "references" into ch5_integrate, ch6_integrate,
    ch7_integrate

```

```
46      when: params.integrate || params.all
47
48      shell:
49      '''
50      #!/bin/bash
51      export PATH="!{params.envPath_integrate}:$PATH"
52
53      mkdir references && cd "$_"
54      wget https://genome-idx.s3.amazonaws.com/bt/GRCh38_noalt_as.
        zip
55      unzip GRCh38_noalt_as.zip
56      rm GRCh38_noalt_as.zip
57      gdown "https://drive.google.com/uc?export=download&confirm=
        qgOc&id=18SUV1abrk_MhYGOG6kzJPIeIJ5Zs5Yvb"
58      gdown "https://drive.google.com/uc?export=download&confirm=
        qgOc&id=14VCiEYWCl5m9bo_tsvNGQDUNUgtLje9Y"
59      tar -xvf INTEGRATE.0.2.6.tar.gz
60      rm INTEGRATE.0.2.6.tar.gz
61      cd INTEGRATE_0_2_6 && mkdir INTEGRATE-build && cd "$_"
62      cmake ../Integrate/ -DCMAKE_BUILD_TYPE=release
63      make
64      '''
65  }
66  process integrate_builder{
67      tag "Building"
68
69      publishDir "${params.outdir}/integrate/references", mode: '
        copy'
70
71      input:
72      val x from ch1_integrate_bwts
73      file refgen from refgen_integrate_builder.mix(
        refgen_integrate_builder_down)
74      file integrate_db from ch4_integrate.mix(ch7_integrate)
75
76      output:
77      file "bwts" into ch8_integrate
78
79      when: params.integrate || params.all
80
81      shell:
82      '''
83      #!/bin/bash
84      export PATH="!{params.envPath_integrate}:$PATH"
85
86      LD_LIBRARY_PATH=/usr/local/lib
87      LD_LIBRARY_PATH=$LD_LIBRARY_PATH:!{integrate_db}/
        INTEGRATE_0_2_6/INTEGRATE-build/vendor/src/libdivsufsort-2.0.1-
        build/lib/
```

```
88      export LD_LIBRARY_PATH
89      mkdir ./bwts
90      !{integrate_db}/INTEGRATE_0_2_6/INTEGRATE-build/bin/Integrate
        mkbwt !{refgen}
91      '''
92  }
93  process integrate_converter{
94      tag "${pair_id}"
95
96      publishDir "${params.outdir}/integrate", mode: 'copy'
97
98      input:
99      tuple pair_id, file(rna_reads), file(integrate_db), file(
        refgen), file(index), file(wgstinput), file(wgsninput) from
        rna_reads_integrate.combine(ch2_integrate.mix(ch5_integrate)).
        combine(refgen_integrate_converter.mix(
        refgen_integrate_converter_down)).combine(refgen_index.mix(
        refgen_index_down)).join(dna_reads_tumor_integrate).join(
        dna_reads_normal_integrate)
100
101     output:
102     tuple pair_id, file("input/${pair_id}") into integrate_input
103
104     when: params.integrate || params.all
105
106     script:
107     """
108     #!/bin/bash
109     export PATH='${params.envPath_integrate}:$PATH'
110
111     tophat --no-coverage-search ${integrate_db}/GRCh38_noalt_as/
        GRCh38_noalt_as ${rna_reads}
112     mkdir input && mkdir input/${pair_id}
113     cp tophat_out/accepted_hits.bam input/${pair_id}
114     cp tophat_out/unmapped.bam input/${pair_id}
115     if ${params.dnabam}; then
116       if ${integrateWGSt}; then
117         cp ${wgstinput} input/${pair_id}/dna.tumor.bam
118       fi
119       if ${integrateWGSn}; then
120         cp ${wgsninput} input/${pair_id}/dna.normal.bam
121       fi
122     elif ${integrateWGSt} || ${integrateWGSn}; then
123       mkdir index_dir
124       cp ${index}/* index_dir
125       if ${integrateWGSt}; then
126         bwa mem index_dir/hg38.fa ${wgstinput} | samtools sort -o
        input/${pair_id}/dna.tumor.bam
127       fi
```

```
128       if ${integrateWGSn}; then
129         bwa mem index_dir/hg38.fa ${wgsninput} | samtools sort -o
    input/${pair_id}/dna.normal.bam
130       fi
131     fi
132     """
133 }
134 process integrate{
135     tag "${pair_id}"
136 
137     publishDir "${params.outdir}/integrate", mode: 'copy'
138 
139     input:
140     tuple pair_id, file(input), file(integrate_db), file(refgen),
    file(bwts) from integrate_input.combine(ch3_integrate.mix(
    ch6_integrate)).combine(refgen_integrate.mix(
    refgen_integrate_down)).combine(ch2_integrate_bwts.mix(
    ch8_integrate))
141 
142     output:
143     file "output/${pair_id}" optional true into integrate_fusions
144 
145     when: params.integrate || params.all
146 
147     shell:
148     '''
149     #!/bin/bash
150     export PATH="!{params.envPath_integrate}:$PATH"
151 
152     cp !{input}/* .
153     parallel samtools index ::: *.bam
154     LD_LIBRARY_PATH=/usr/local/lib
155     LD_LIBRARY_PATH=$LD_LIBRARY_PATH:!{integrate_db}/
    INTEGRATE_0_2_6/INTEGRATE-build/vendor/src/libdivsufsort-2.0.1-
    build/lib/
156     export LD_LIBRARY_PATH
157     !{integrate_db}/INTEGRATE_0_2_6/INTEGRATE-build/bin/Integrate
    fusion !{refgen} !{integrate_db}/annot.refseq.txt !{bwts}
    accepted_hits.bam unmapped.bam !{command1} !{command2}
158     mkdir output && mkdir output/!{pair_id}
159     cp *.tsv output/!{pair_id}
160     cp *.txt output/!{pair_id}
161     '''
162 }
```

Listing 5.5: Integrate processes.

Chapter 6

Results and discussion

The first part of this chapter is dedicated to describing the files used to test the FusionFlow pipeline. Subsequently, the pipeline output files and results are reported and discussed.

6.1 Files

Simulated trial files were used to perform the tests. These files are publicly available (`https://github.com/ndaniel/fusioncatcher/tree/master/test`) and can also be found in the FusionFlow GitHub repository. Moreover, these files are used in test_docker and test_local profiles to verify the correct functioning of the pipeline. The test files are two Fastq compressed paired-end files. The short reads were selected manually such that they cover 17 already known fusion genes:

- FGFR3 - TACC3
- FIP1L1 - PDGFRA
- GOPC - ROS1
- IGH - CRLF2
- HOOK3 - RET
- AKAP9 - BRAF
- EWSR1 - ATF1

- TMPRSS2 - ETV1
- EWSR1 - FLI1
- ETV6 - NTRK3
- BRD4 - NUTM1
- CD74 - ROS1
- CIC - DUX4
- DUX4 - IGH
- EML4 - ALK
- MALT1 - IGH
- NPM1 - ALK

6.2 Tests and results

The tests were performed in two different conditions: on the Polito Philae server and the Polito Philae server inside a Docker container.
In the first case, the Conda virtual environments were manually created on the server. The environments paths, the test files, and the local profile were specified in the command line to perform the test.
In the second case, the Conda virtual environments were created inside the Docker container directly by the Dockerfile shown in appendix A. The default environments paths already point to the Docker environments. Thus, it is not needed to specify the environments paths in the command line in this case. The test files and the local profile were specified in the command line to execute this test.
In both test cases, the outputs obtained were satisfactory. The outputs obtained in the two cases were the same and are shown in Figure 6.1, 6.2, 6.3, 6.4, 6.5.

```
#gene1  gene2   strand1(gene/fusion)    strand2(gene/fusion)    breakpoint1     breakpoint2     site1
site2   type    split_reads1    split_reads2    discordant_mates        coverage1       coverage2
confidence      reading_frame   tags    retained_protein_domains
        closest_genomic_breakpoint1     closest_genomic_breakpoint2     gene_id1        gene_id2
transcript_id1  transcript_id2  direction1      direction2      filters fusion_transcript
peptide_sequence        read_identifiers
FGFR3   TACC3   +/+     +/+     4:1806934       4:1727977       CDS/splice-site CDS     duplication
91      106     300     248     444     high    out-of-frame    Mitelman        Immunoglobulin_I-
set_domain(100%),Immunoglobulin_domain(100%)|  .       .       ENSG000000680
78 ENSG00000013810 ENST00000440486 ENST00000313288 downstream      upstream
duplicates(109),low_entropy(8),mismappers(2),mismatches(10) […]
FIP1L1  PDGFRA  +/+     +/+     4:53425965      4:54274925      CDS/splice-site CDS     deletion
34      78      16      45      126     high    .       .       .       .       .       ENSG00000145216
ENSG00000134853 ENST00000505125 ENST00000257290 downstream      upstream
duplicates(21),inconsistently_clipped(1),low_entropy(1),mismatches(1) […]
GOPC    ROS1    -/-     -/-     6:117566854     6:117321394     CDS/splice-site CDS/splice-site
deletion/read-through   35      16      25      123     145     high    in-frame        .       .
.       .       ENSG00000047932 ENSG00000047936 ENST00000052569 ENST00000368508 upstream
downstream      duplicates(77),low_entropy(5),mismatches(1) […]
GOPC    ROS1    -/-     -/-     6:117566854     6:117320030     CDS/splice-site CDS/splice-site
deletion/read-through   1       0       8       123     29      low     out-of-frame    .       .
.       .       ENSG00000047932 ENSG00000047936 ENST00000052569 ENST00000368508 upstream
downstream      mismatches(1) […]
HOOK3   RET     +/+     +/+     8:42968214      10:43116584     CDS/splice-site CDS/splice-site
translocation   9       3       2       35      8       high    in-frame        .       .       .
.       ENSG00000168172 ENSG00000165731 ENST00000307602 ENST00000355710 downstream      upstream
. […]
ETV6    NTRK3   +/+     -/-     12:11869969     15:87940753     CDS/splice-site CDS/splice-site
translocation   6       4       2       87      7       high    in-frame        .       .       .
.       ENSG00000139083 ENSG00000140538 ENST00000396373 ENST00000394480 downstream      downstream
mismatches(1) […]
AKAP9   BRAF    +/+     -/-     7:92003235      7:140787584     CDS/splice-site CDS/splice-site
inversion       3       5       4       51      35      high    in-frame        .       .       .
.       ENSG00000127914 ENSG00000157764 ENST00000356239 ENST00000496384 downstream      downstream
duplicates(1),mismatches(2) […]
BRD4    NUTM1   -/-     +/+     19:15254152     15:34347969     CDS/splice-site CDS/splice-site
translocation   2       5       1       101     11      high    in-frame        .       .       .
.       ENSG00000141867 ENSG00000184507 ENST00000360016 ENST00000614490 upstream        upstream
. […]
EML4    ALK     +/+     -/-     2:42301394      2:29223584      intron  5'UTR   inversion       1
4       3       71      8       high    in-frame        .       .       .       .       ENSG00000143924
ENSG00000171094 ENST00000318522 ENST00000642122 downstream      downstream      mismatches(2) […]
CD74    ROS1    -/-     -/-     5:150404680     6:117324415     CDS/splice-site CDS/splice-site
translocation   2       5       0       8       6       high    in-frame        .       .       .
.       ENSG00000019582 ENSG00000047936 ENST00000353334 ENST00000368508 upstream        downstream
low_entropy(1) […]
NPM1    ALK     +/+     -/-     5:171391799     2:29223528      CDS/splice-site CDS/splice-site
translocation   0       5       2       0       51      high    in-frame        .       .       .
.       ENSG00000181163 ENSG00000171094 ENST00000517671 ENST00000389048 downstream      downstream
. […]
TMPRSS2 ETV1    -/-     -/-     21:41494375     7:13935838      CDS     CDS     translocation   4
8       3       8       115     medium  in-frame        .       |Ets-domain(100%)       .       .
ENSG00000184012 ENSG00000006468 ENST00000454499 ENST00000242066 upstream        downstream      . […]
EWSR1   ATF1    +/+     +/+     22:29287134     12:50814280     CDS/splice-site CDS/splice-site
translocation   4       4       1       1094    145     medium  in-frame        .       .       .
.       ENSG00000182944 ENSG00000123268 ENST00000629659 ENST00000262053 downstream      upstream
mismatches(1) […]
CIC     DUX4L8  +/+     +/+     19:42295049     4:190068609     CDS     exon    translocation   4
2       3       10      6       medium  out-of-frame    .       .       .       .       ENSG00000079432
ENSG00000281720 ENST00000572681 ENST00000629013 downstream      upstream
duplicates(3),multimappers(6)   […]
EWSR1   FLI1    +/+     +/+     22:29287134     11:128807180    CDS/splice-site CDS/splice-site
translocation   2       3       3       1094    29      medium  in-frame        .       .       .
.       ENSG00000182944 ENSG00000151702 ENST00000629659 ENST00000344954 downstream      upstream
mismatches(2) […]
```

Figure 6.1: *fusions.tsv* Arriba output file (peptide sequence and read identifiers are not shown).

```
GeneName1       GeneName2       chr1    Breakpoint1     strand1 chr2    Breakpoint2     strand2
EnsemblGene1    EnsemblGene2    crossingreads   spanningreads   mean.insertsize homology        fusiontype
Blacklist       InfoGene1       InfoGene2       JunctionSequence
        GeneExpr1       GeneExpr2       GeneExpr_Fused  ES      GJS     US      EricScore
EWSR1   ATF1    22      29287134        +       12      50814280        +       ENSG00000182944
ENSG00000123268 11      25      71.12           inter-chromosomal               EWS RNA binding protein 1
[Source:HGNC Symbol;Acc:HGNC:3508]    activating transcription factor 1 [Source:HGNC Symbol;Acc:HGNC:783]
[…]     37      12.27   553.3   0.9208  0.6225  0.44    0.914900173537433
EWSR1   FLI1    22      29287134        +       11      128807180       +       ENSG00000182944
ENSG00000151702 9       12      82.09           inter-chromosomal               EWS RNA binding protein 1
[Source:HGNC Symbol;Acc:HGNC:3508]    Fli-1 proto-oncogene, ETS transcription factor [Source:HGNC
Symbol;Acc:HGNC:3749] […]     37      3.29    600.14  0.9193  0.7441  0.75    0.903905557453987
AKAP9   BRAF    7       92003235        +       7       140787584       -       ENSG00000127914
ENSG00000157764 11      24      78.53           intra-chromosomal               A-kinase anchoring protein
9 [Source:HGNC Symbol;Acc:HGNC:379]  B-Raf proto-oncogene, serine/threonine kinase [Source:HGNC
Symbol;Acc:HGNC:1097] […]     9.56    9.72    37.9    0.9222  0.5874  0.458333333333333
0.981171578900953
BRD4    NUTM1   19      15254152        -       15      34347969        +       ENSG00000141867
ENSG00000184507 7       13      73.2            inter-chromosomal               bromodomain containing 4
[Source:HGNC Symbol;Acc:HGNC:13575]    NUT midline carcinoma, family member 1 [Source:HGNC
Symbol;Acc:HGNC:29919] […]     4.19    17.98   83.9    0.9081  0.7745  0.538461538461538
0.978955922397591
EML4    ALK     2       42301392        +       2       29223531        -       ENSG00000143924
ENSG00000171094 7       2       76.33           intra-chromosomal               echinoderm microtubule
associated protein like 4 [Source:HGNC Symbol;Acc:HGNC:1316]     anaplastic lymphoma receptor tyrosine
kinase [Source:HGNC Symbol;Acc:HGNC:427] […]     13.57   8.01    77.66   -0.0241 0.1302  0.285714285714286
0.160394590429103
TMPRSS2 ETV1    21      41494380        -       7       13935843        -       ENSG00000184012
ENSG00000006468 9       16      77.59           inter-chromosomal               transmembrane protease,
serine 2 [Source:HGNC Symbol;Acc:HGNC:11876]    ets variant 1 [Source:HGNC Symbol;Acc:HGNC:3490] […]    0
17.69   18.49   0.9481  0.6946  0.5625  0.964701061516341
CD74    ROS1    5       150404680       -       6       117324415       -       ENSG00000019582
ENSG00000047936 3       14      46.53           inter-chromosomal               CD74 molecule, major
histocompatibility complex, class II invariant chain [Source:HGNC Symbol;Acc:HGNC:1697]    ROS proto-
oncogene 1 , receptor tyrosine kinase [Source:HGNC Symbol;Acc:HGNC:10261] […]     2.44    2.11    19.87
0.9159  0.567   0.214285714285714       0.950074467523889
HOOK3   RET     8       42968214        +       10      43116584        +       ENSG00000168172
ENSG00000165731 12      21      86.49           inter-chromosomal               hook microtubule-tethering
protein 3 [Source:HGNC Symbol;Acc:HGNC:23576]        ret proto-oncogene [Source:HGNC Symbol;Acc:HGNC:9967]
[…]     2.44    3.31    31.82   0.9177  0.5627  0.571428571428571       0.978640641009045
ETV6    NTRK3   12      11869970        +       15      87940752        -       ENSG00000139083
ENSG00000140538 7       24      85.12           inter-chromosomal               ets variant 6 [Source:HGNC
Symbol;Acc:HGNC:3495]        neurotrophic tyrosine kinase, receptor, type 3 [Source:HGNC
Symbol;Acc:HGNC:8033] […]     9.48    0.45    54.74   0.8715  0.5284  0.291666666666667
0.913070827796455
FGFR3   TACC3   4       1806934 +       4       1727977 +       ENSG00000068078 ENSG00000013810 978
378     208.89          Read-Through            fibroblast growth factor receptor 3 [Source:HGNC
Symbol;Acc:HGNC:3690]  transforming, acidic coiled-coil containing protein 3 [Source:HGNC
Symbol;Acc:HGNC:11524] […]     0       1.58    101.36  0.9004  0.8863  0.386503067484663
0.981171578900953
FIP1L1  PDGFRA  4       53425965        +       4       54274925        +       ENSG00000145216
ENSG00000134853 68      39      253.23          intra-chromosomal               factor interacting with
PAPOLA and CPSF1 [Source:HGNC Symbol;Acc:HGNC:19124]    platelet-derived growth factor receptor, alpha
polypeptide [Source:HGNC Symbol;Acc:HGNC:8803] […]     0       0.01    29.15   0.7565  0.668
0.573529411764706       0.982698223970533
GOPC    ROS1    6       117566854       -       6       117321394       -       ENSG00000047932
ENSG00000047936 140     73      149.07          intra-chromosomal               golgi-associated PDZ and
coiled-coil motif containing [Source:HGNC Symbol;Acc:HGNC:17643]       ROS proto-oncogene 1 , receptor
tyrosine kinase [Source:HGNC Symbol;Acc:HGNC:10261] […]     0       2.11    26.21   0.9235  0.6472
0.521428571428571       0.982698223970533
CIC     DUX4    19      42295048        +       4       190174447       +       ENSG00000079432
ENSG00000260596 6       6       175.89          inter-chromosomal               capicua transcriptional
repressor [Source:HGNC Symbol;Acc:HGNC:14214]   double homeobox 4 [Source:HGNC Symbol;Acc:HGNC:50800] […]
0       1.34    5.54    0.9347  0.6851  1       0.692371740568776
```

Figure 6.2: *MyEric.results.total.tsv* EricScript output file (junction sequences are not shown).

```
Very short summary of found candidate somatic fusion genes
========================================================== ====

Found 17 somatic fusion gene(s), which are as follows:
  * FGFR3--TACC3  (already known fusion)
  * FIP1L1--PDGFRA  (already known fusion)
  * GOPC--ROS1  (already known fusion; exon-exon fusion junction)
  * IGH@--CRLF2  (already known fusion)
  * HOOK3--RET  (already known fusion; exon-exon fusion junction)
  * AKAP9--BRAF  (already known fusion; exon-exon fusion junction)
  * EWSR1--ATF1  (already known fusion; exon-exon fusion junction)
  * TMPRSS2--ETV1  (already known fusion)
  * EWSR1--FLI1  (already known fusion; exon-exon fusion junction)
  * ETV6--NTRK3  (already known fusion; exon-exon fusion junction)
  * BRD4--NUTM1  (already known fusion; exon-exon fusion junction)
  * CD74--ROS1  (already known fusion; exon-exon fusion junction)
  * CIC--DUX4  (already known fusion)
  * DUX4--IGH@  (already known fusion)
  * EML4--ALK  (already known fusion)
  * MALT1--IGH@  (already known fusion)
  * NPM1--ALK  (already known fusion; exon-exon fusion junction)

Found 22 somatic fusion transcript(s).
```

Figure 6.3: *summary_candidate_fusions.txt* FusionCatcher summary output file.

```
Fusion_Candidate        5_Prime 3_Prime Reciprocal      Tier    Type    EN_RNA  SP_RNA  EN_DNA_T
SP_DNA_T        Splicings
1       EWSR1   ATF1    N       1       Inter_Chromosomal       11      8       11      8
EWSR1>>ATF1(Inter_Chromosomal Canonical 8);
2       BRD4    NUTM1   N       1       Inter_Chromosomal       2       7       7       5
BRD4>>NUTM1(Inter_Chromosomal Canonical 7);
3       EWSR1   FLI1    N       1       Inter_Chromosomal       3       5       9       2
EWSR1>>FLI1(Inter_Chromosomal Canonical 5);
4       ETV6    NTRK3   N       1       Inter_Chromosomal       2       5       8       5
ETV6>>NTRK3(Inter_Chromosomal Canonical 1);ETV6>>NTRK3(Inter_Chromosomal Canonical
2);ETV6>>NTRK3(Inter_Chromosomal Canonical 2);
5       NPM1    ALK     N       1       Inter_Chromosomal       2       5       3       3
NPM1>>ALK(Inter_Chromosomal Canonical 5);
6       GOPC    ROS1    N       1       Intra_Chromosomal       78      68      139     19
GOPC>>ROS1(Intra_Chromosomal Canonical 68);
7       AKAP9   BRAF    N       1       Intra_Chromosomal       4       3       7       4
AKAP9>>BRAF(Intra_Chromosomal Canonical 3);
8       TMPRSS2 ETV1    N       4       Inter_Chromosomal       2       10      9       8
TMPRSS2>>ETV1(Inter_Chromosomal 10);
9       FGFR3   TACC3   N       4       Intra_Chromosomal       516     214     978     117
FGFR3>>TACC3(Intra_Chromosomal 214);
10      FIP1L1  PDGFRA  N       4       Intra_Chromosomal       10      25      45      22
FIP1L1>>PDGFRA(Intra_Chromosomal 25);
11      CIC     DBET    N       6       Inter_Chromosomal       2       3       0       0
CIC>>DBET(Inter_Chromosomal 3);
```

Figure 6.4: *summary.tsv* Integrate summary output file.

```
#Fusion: NPM1_ENST00000296930.9:exon:4|+chr5:171391798___ALK_ENST00000389048.7:exon:20|-chr2:29223527
(total: 3, unique:2)
>1, break:56, diff:(0 0), read direction: original direction, name: 9O1
CTTAAGGTTGAAGTGTGGTTCAGGGCCAGTGCATATTAGTGGACAGCACTTAGTAG TGTACCGCCGGAAGCACCAGGAGCTGCAAGCCATGCAGATGGAGC
>2, break:56, diff:(0 0), read direction: original direction, name: 9O2
CTTAAGGTTGAAGTGTGGTTCAGGGCCAGTGCATATTAGTGGACAGCACTTAGTAG TGTACCGCCGGAAGCACCAGGAGCTGCAAGCCATGCAGATGGAGC
>3, break:48, diff:(0 0), read direction: original direction, name: 9O0
TGAAGTGTGGTTCAGGGCCAGTGCATATTAGTGGACAGCACTTAGTAG TGTACCGCCGGAAGCACCAGGAGCTGCAAGCCATGCAGATGGAGCTGCAGAGC

#Fusion: ROS1_ENST00000368508.7:intron:34|+chr6:117321395___GOPC_ENST00000052569.10:exon:7|+chr6:117566855
(total: 61, unique:26)
>1, break:113, diff:(0 1), read direction: original direction, name: 9HL merged_diff_0
GGCCTACTCCGGCTGCCAGACCTCGCAGCTCAGCCAACTCTTTGTCTTCGTTTATAAGCACTGTCACCCCTTCCTTGGCACTTTTTTGATTCTTTAATCTTCTATGC
CAGACT TGTAATACTTTGATTTCCCCACTTGTGTCTTTGCAACTAGCGCCAG
>2, break:112, diff:(0 0), read direction: original direction, name: 9HS merged_diff_1
GCCTACTCCGGCTGCCAGACCTCGCAGCTCAGCCAACTCTTTGTCTTCGTTTATAAGCACTGTCACCCCTTCCTTGGCACTTTTTTGATTCTTTAATCTTCTATGCC
AGACT TGTAATACTTTGATTTCCCCACTTGTGTCTTTGCAACTAGCACCAGGGTTACCACCTCCT
>3, break:112, diff:(0 0), read direction: original direction, name: 9JJ merged_diff_1
GCCTACTCCGGCTGCCAGACCTCGCAGCTCAGCCAACTCTTTGTCTTCGTTTATAAGCACTGTCACCCCTTCCTTGGCACTTTTTTGATTCTTTAATCTTCTATGCC
AGACT TGTAATACTTTGATTTCCCCACTTGTGTCTTTGCAACTAGCACCAGGGTTACCACCTCCT
[…]

#Fusion: ETV6_ENST00000396373.8:exon:5|+chr12:11869968___NTRK3_ENST00000394480.6:exon:15|-chr15:87940752
(total: 4, unique:4)
>1, break:95, diff:(0 0), read direction: original direction, name: 6MX merged_diff_0
CCTCTCTCATCGGGAAGACCTGGCTTACATGAACCACATCATGGTCTCTGTCTCCCCGCCTGAAGAGCACGCCATGCCCATTGGGAGAATAGCAG
ATGTGCAGCACATTAAGAGGAGAGACATCGTGCTGAAGCGAGAACTGGG
>2, break:68, diff:(0 1), read direction: original direction, name: 3L1 merged_diff_0
CATGAACCACATCATGGTCTCTGTCTCCCCGCCTGAAGAGCACGCCATGCCCATTGGGAGAATAGCAG
ATGTGCAGCACATTAAGAGGAGAGACATCGTGCTGAAGCGAGAACTGGGG
>3, break:59, diff:(1 0), read direction: reversed complement, name: 57B
NATCATGGTCTCTGTCTCCCCGCCTGAAGAGCACGCCATGCCCATTGGGAGAATAGCAG ATGTGCAGCACATTAAGAGGAGAGACATCGTGCTGAAGCGAG
[…]

# references/genefuse -r hg38.fa -f references/druggable.hg38.csv -1 reads_3.fq.gz -2 reads_4.fq.gz -h
report.html
# genefuse v0.6.1, time used: 128 seconds
```

Figure 6.5: *result* GeneFuse output file.

Each gene fusion discovery tool gives as output one or more files in specific formats. Generally, a summary file is also produced in output to allow a quick predictions overview.

The outputs obtained from the tools are concordant with the gene fusions previously specified. All the tools in the pipeline recognize at least ten predictions out of seventeen fusions, except for GeneFuse that recognizes just three of them. This aspect is related to the different gene fusions discovery algorithm used for DNA data instead of RNA data.

In order to select the final gene fusions prediction drivers, different approaches can be used. The typical practice is to use the union or intersection of tools predictions. The union of the results gives a numerous set of predictions. This approach increases the probability of including the real drivers of cancer processes. However, it enhances the possibilities to incorporate also false positives or passenger

mutations. Using the intersection approach, conversely, decreases the number of predictions radically. This approach allows discarding false positives and passenger mutations. However, this selection could also cause the discarding of the cancer drivers.

In this test case, the union of the results contains nineteen gene fusions predictions, while the intersection includes just two of them (ETV6-NTRK3 and GOPC-ROS1). The tests performed were used to validate the pipeline and can be repeated in test_docker and test_local profiles to check the correct functioning of the pipeline.

Chapter 7

Conclusions and future works

In this thesis work, a novel pipeline for gene fusions discovery was designed. FusionFlow is an easy-to-use, flexible, highly reproducible, and integrated pipeline. The pipeline includes five gene fusion discovery tools that take as input both RNA and DNA data. Docker and Conda technologies allow performing tools installations, avoiding version conflicts. In addition, the Nextflow pipelining tool allows the execution of the five tools in parallel, optimizing time and resources usage and managing the tools installations and the files allocation. The pipeline was tested using publicly available test files. The tests were performed using a local profile in two conditions: on the Polito Philae server and the Polito Philae server inside a docker container. In both cases, the outputs were satisfactory. Thus, the FusionFlow pipeline is available for further validation over real DNA and RNA genomic data.

This work represents a foundation on which improvements and future works can be built. Indeed, one of the main problems related to gene fusion discovery is determining which gene fusions are drivers of cancer processes and not just passenger mutations. The fusion detection tools already provide a first step for the solution of this problem. Indeed, fusion detection tools filter the candidate gene fusions based on the sample's reads, trying to decrease as much as possible the number of false positives. However, generally, this step is insufficient to determine the cancer drivers. Thus, a second step is required. This step is provided by

specific post-processing tools, known as prioritization tools, that predict a gene fusion's oncogenic potential. There is a high number of prioritization tools such as Oncofuse [25], and Pegasus [26]. These tools are based on machine learning (ML) algorithms trained with the protein domains of the fusion proteins and allow the selection of the most probable cancer drivers. The post-processing step could also be completed by adding a further algorithm. This algorithm performs comparisons between the outputs of the tool and selects the more probable driver of cancer processes analyzing the union and the intersection and taking into account the different characteristics of the gene fusion detection tools.

Another crucial question is the one related to the visualization tools. Humans are efficient in distinguishing true positives from false positives if the evidence is provided in an easily interpretable form. These tools also allow better to interpret the potential consequence of gene fusions events. Several visualization tools were released in the last years, such as INTEGRATE-vis [27], FGviewer [28], and FuSpot [29].

Appendix A

Dockerfile

```
FROM continuumio/anaconda3

LABEL description="Docker image containing all requirements for genefusion pipeline"

RUN apt-get update
RUN apt-get install -y wget
RUN apt-get install unzip
RUN apt-get install make
RUN apt-get install -y build-essential
RUN apt-get install -y python2
RUN apt-get install -y zlib1g-dev
RUN apt-get install -y libtbb-dev libtbb2 libc6-dev
RUN apt-get install -y libncurses5-dev

RUN apt-get install -y locales
RUN localedef -i en_US -c -f UTF-8 -A /usr/share/locale/locale.alias en_US.UTF-8

RUN pip install gdown

# Install Nextflow
RUN apt update && apt -y install default-jre && apt -y install openjdk-11-jre-headless
RUN curl -s https://get.nextflow.io | bash
RUN cd  /home/ && curl -s https://get.nextflow.io | bash

# Install the conda environment for EricScript
COPY environment_ericscript.yml /
RUN conda env create -f /environment_ericscript.yml && conda clean -a

# Install the conda environment for Arriba
```

```
30 COPY environment_arriba.yml /
31 RUN conda env create -f /environment_arriba.yml && conda clean -a
32
33 # Install the conda environment for FusionCatcher
34 COPY environment_fusioncatcher.yml /
35 RUN conda env create -f /environment_fusioncatcher.yml && conda
       clean -a
36
37 RUN wget https://github.com/ndaniel/fastqtk/archive/refs/tags/v0
       .27.zip
38 RUN unzip v0.27.zip && rm v0.27.zip
39 WORKDIR fastqtk-0.27
40 RUN make
41 RUN mv fastqtk /opt/conda/envs/fusioncatcher/bin/ && cd ..
42 WORKDIR /home
43
44 # Install the conda environment for INTEGRATE
45 COPY environment_integrate.yml /
46 RUN conda env create -f /environment_integrate.yml && conda clean
       -a
47
48 RUN wget https://ccb.jhu.edu/software/tophat/downloads/tophat
       -2.1.1.Linux_x86_64.tar.gz
49 RUN tar -xvzf tophat-2.1.1.Linux_x86_64.tar.gz
50 RUN rm tophat-2.1.1.Linux_x86_64.tar.gz && rm tophat-2.1.1.
       Linux_x86_64/tophat
51 RUN cp -r tophat-2.1.1.Linux_x86_64/* /opt/conda/envs/integrate/
       bin/
52
53 RUN gdown "https://drive.google.com/uc?export=download&confirm=
       qgOc&id=1A4JyTwjnwqDjWqVuEgt1sfDQrwU3oNbv"
54 RUN chmod +x tophat
55 RUN mv tophat /opt/conda/envs/integrate/bin/
56
57 # Install the conda environment for GeneFuse
58 COPY environment_genefuse.yml /
59 RUN conda env create -f /environment_genefuse.yml && conda clean -
       a
```

Bibliography

[1] Anna Williford and Esther Betrán. *Gene Fusion*. May 2013. DOI: 10.1002/9780470015902.a0005099.pub3. URL: https://onlinelibrary.wiley.com/doi/10.1002/9780470015902.a0005099.pub3 (cit. on pp. 11, 15).

[2] Sebastian Uhrig et al. *Accurate and efficient detection of gene fusions 1 from RNA sequencing data Corresponding authors 20 Keywords* (cit. on pp. 12, 29).

[3] Somak Roy et al. *Standards and Guidelines for Validating Next-Generation Sequencing Bioinformatics Pipelines: A Joint Recommendation of the Association for Molecular Pathology and the College of American Pathologists*. Jan. 2018. DOI: 10.1016/j.jmoldx.2017.11.003 (cit. on p. 12).

[4] Jin Zhang, Nicole M. White, Heather K. Schmidt, Robert S. Fulton, Chad Tomlinson, Wesley C. Warren, Richard K. Wilson, and Christopher A. Maher. «INTEGRATE: Gene fusion discovery using whole genome and transcriptome data». In: *Genome Research* 26 (1 Jan. 2016), pp. 108–118. ISSN: 15495469. DOI: 10.1101/gr.186114.114 (cit. on pp. 12, 31).

[5] Qingguo Wang, Junfeng Xia, Peilin Jia, William Pao, and Zhongming Zhao. «Application of next generation sequencing to human gene fusion detection: Computational tools, features and perspectives». In: *Briefings in Bioinformatics* 14 (4 July 2013), pp. 506–519. ISSN: 14675463. DOI: 10.1093/bib/bbs044 (cit. on p. 14).

[6] Natasha S. Latysheva and M. Madan Babu. «Discovering and understanding oncogenic gene fusions through data intensive computational approaches». In: *Nucleic Acids Research* 44 (10 June 2016), pp. 4487–4503. ISSN: 13624962. DOI: 10.1093/nar/gkw282 (cit. on p. 15).

[7] Yue Wang, Tao Shi, Xueru Song, Baorui Liu, and Jia Wei. *Gene fusion neoantigens: Emerging targets for cancer immunotherapy*. May 2021. DOI: 10.1016/j.canlet.2021.02.023 (cit. on pp. 16, 18).

[8] The Luigi Authors. *Luigi*. URL: https://luigi.readthedocs.io/en/stable/ (cit. on p. 22).

[9] Centre for Genomic Regulation (CRG). *Nextflow Documentation*. URL: https://www.nextflow.io/docs/latest/index.html (cit. on pp. 22, 24, 25).

[10] Guillaume Theaud, Jean Christophe Houde, Arnaud Boré, François Rheault, Felix Morency, and Maxime Descoteaux. «TractoFlow: A robust, efficient and reproducible diffusion MRI pipeline leveraging Nextflow Singularity». In: *NeuroImage* 218 (Sept. 2020). ISSN: 10959572. DOI: 10.1016/j.neuroimage.2020.116889 (cit. on p. 22).

[11] Alexander Kropp and Roberto Torre. *Docker: containerize your application*. 2020. DOI: 10.1016/b978-0-12-820488-7.00026-8 (cit. on p. 26).

[12] *Docker Official Website* (cit. on p. 26).

[13] Alexander Dobin, Carrie A. Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R. Gingeras. «STAR: Ultrafast universal RNA-seq aligner». In: *Bioinformatics* 29 (1 Jan. 2013), pp. 15–21. ISSN: 13674803. DOI: 10.1093/bioinformatics/bts635 (cit. on pp. 29–31).

[14] Heng Li and Richard Durbin. «Fast and accurate short read alignment with Burrows-Wheeler transform». In: *Bioinformatics* 25 (14 July 2009), pp. 1754–1760. ISSN: 13674803. DOI: 10.1093/bioinformatics/btp324 (cit. on p. 29).

[15] W. J. Kent. «BLAT—The BLAST-Like Alignment Tool». In: *Genome Research* 12 (4 Mar. 2002), pp. 656–664. ISSN: 1088-9051. DOI: 10.1101/gr.229202 (cit. on pp. 29, 30).

[16] Matteo Benelli, Chiara Pescucci, Giuseppina Marseglia, Marco Severgnini, Francesca Torricelli, and Alberto Magi. «Discovering chimeric transcripts in paired-end RNA-seq data by using EricScript». In: *Bioinformatics* 28 (24 Dec. 2012), pp. 3232–3239. ISSN: 13674803. DOI: 10.1093/bioinformatics/bts617 (cit. on p. 29).

[17] Paul Flicek et al. «Ensembl 2013». In: *Nucleic Acids Research* 41 (D1 Jan. 2013). ISSN: 03051048. DOI: 10.1093/nar/gks1236 (cit. on p. 30).

[18] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L. Salzberg. «Ultrafast and memory-efficient alignment of short DNA sequences to the human genome». In: *Genome Biology* 10 (3 Mar. 2009). ISSN: 14747596. DOI: 10.1186/gb-2009-10-3-r25 (cit. on p. 30).

[19] Ben Langmead and Steven L. Salzberg. «Fast gapped-read alignment with Bowtie 2». In: *Nature Methods* 9 (4 Apr. 2012), pp. 357–359. ISSN: 15487091. DOI: 10.1038/nmeth.1923 (cit. on p. 30).

[20] Daniel Nicorici, Mihaela Satalan, Henrik Edgren, Sara Kangaspeska, Astrid Murumagi, Olli Kallioniemi, Sami Virtanen, and Olavi Kilkku. «FusionCatcher - a tool for finding somatic fusion genes in paired-end RNA-sequencing data». In: *bioRxiv* (2014), p. 011650. DOI: 10.1101/011650 (cit. on p. 30).

[21] Shifu Chen, Ming Liu, Tanxiao Huang, Wenting Liao, Mingyan Xu, and Jia Gu. «Genefuse: Detection and visualization of target gene fusions from DNA sequencing data». In: *International Journal of Biological Sciences* 14 (8 May 2018), pp. 843–848. ISSN: 14492288. DOI: 10.7150/ijbs.24626 (cit. on p. 31).

[22] Thomas D. Wu and Serban Nacu. «Fast and SNP-tolerant detection of complex variants and splicing in short reads». In: *Bioinformatics* 26 (7 Feb. 2010), pp. 873–881. ISSN: 13674803. DOI: 10.1093/bioinformatics/btq057 (cit. on p. 31).

[23] Daehwan Kim, Geo Pertea, Cole Trapnell, Harold Pimentel, Ryan Kelley, and Steven L Salzberg. *TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions*. 2013. URL: http://ccb.jhu.edu/software/tophat. (cit. on p. 31).

[24] Shahzad Ahmed, M Usman Ali, Javed Ferzund, Muhammad Atif Sarwar, Abbas Rehman, and Atif Mehmood. *Modern Data Formats for Big Bioinformatics Data Analytics*. 2017. URL: www.ijacsa.thesai.org (cit. on pp. 32–34).

[25] Mikhail Shugay, Iñigo Ortiz De Mendíbil, José L. Vizmanos, and Francisco J. Novo. «Oncofuse: A computational framework for the prediction of the oncogenic potential of gene fusions». In: *Bioinformatics* 29 (20 Oct. 2013), pp. 2539–2546. ISSN: 13674803. DOI: 10.1093/bioinformatics/btt445 (cit. on p. 71).

[26] Francesco Abate et al. «Pegasus: A comprehensive annotation and prediction tool for detection of driver gene fusions in cancer». In: *BMC Systems Biology* 8 (1 Sept. 2014). ISSN: 17520509. DOI: 10.1186/s12918-014-0097-z (cit. on p. 71).

[27] Jin Zhang, Teng Gao, and Christopher A. Maher. «INTEGRATE-Vis: A tool for comprehensive gene fusion visualization». In: *Scientific Reports* 7 (1 Dec. 2017). ISSN: 20452322. DOI: 10.1038/s41598-017-18257-2 (cit. on p. 71).

[28] Pora Kim, Ke Yiya, and Xiaobo Zhou. «FGviewer: An online visualization tool for functional features of human fusion genes». In: *Nucleic Acids Research* 48 (1 2021), W313–W320. ISSN: 13624962. DOI: 10.1093/NAR/GKAA364 (cit. on p. 71).

[29] Jackson A. Killian, Taha M. Topiwala, Alex R. Pelletier, David E. Frankhouser, Pearlly S. Yan, and Ralf Bundschuh. «FuSpot: A web-based tool for visual evaluation of fusion candidates». In: *BMC Genomics* 19 (1 Feb. 2018). ISSN: 14712164. DOI: 10.1186/s12864-018-4486-3 (cit. on p. 71).