UNIVERSIDADE NOVA DE LISBOA

MScCBBi

MASTER IN
COMPUTATIONAL BIOLOGY
& BIOINFORMATICS

RODRIGO DE JESUS EUSÉBIO
BSc in Biology

Re-implementation of a data analysis pipeline for the genetic characterization of viral pathogens in the context of laboratory surveillance of viral outbreaks

Sep, 2023

ITQB NOVA
NOVA SCHOOL OF SCIENCE & TECHNOLOGY
NOVA MEDICAL SCHOOL
NOVA IMS Information Management School
DESDE 1902 INSTITUTO DE HIGIENE E MEDICINA TROPICAL UNIVERSIDADE NOVA DE LISBOA

DEPARTMENT OF
LIFE SCIENCES

RE-IMPLEMENTATION OF A DATA ANALYSIS PIPELINE FOR THE GENETIC CHARACTERIZATION OF VIRAL PATHOGENS IN THE CONTEXT OF LABORATORY SURVEILLANCE OF VIRAL OUTBREAKS

RODRIGO DE JESUS EUSÉBIO

BSc in Biology

Adviser: Daniel Sobral
Assistant Researcher, Instituto Nacional de Saúde Doutor Ricardo Jorge

Co-adviser: Ana Rita Grosso
Assistant Professor, NOVA School of Science and Technology

Examination Committee

Chair: Paula Maria Theriaga Mendes Bernardo Gonçalves
Associate Professor, NOVA School of Science and Technology

Rapporteur: Miguel Monsanto Pinheiro
Senior Developer, Critical Techworks

Adviser: Daniel Sobral
Assistant Researcher, Instituto Nacional de Saúde Doutor Ricardo Jorge

MASTER IN COMPUTATIONAL BIOLOGY AND BIOINFORMATICS

NOVA University Lisbon
09, 2023

Re-Implementation of a data analysis pipeline for the genetic characterization of viral pathogens in the context of laboratory surveillance of viral outbreaks

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.0.3) [1].

Acknowledgements

This work was supported by Instituto Nacional de Computação Distribuida (INCD) through project 2022.23037.CPCA.A0. INCD was funded by FCT and FEDER under the project 22153-01/SAICT/2016.

I am deeply thankful to my supervisor, Daniel Sobral, for his unwavering support, guidance, and dedication to ensuring the quality of my work. Your mentorship has been instrumental in shaping this thesis.

I extend my sincere appreciation to Vitor Borges for the countless insightful discussions and suggestions provided throughout the entire thesis journey.

I am grateful to Professor Ana Rita Grosso for her pivotal role in guiding me in selecting the research theme and for her invaluable assistance during the final phase of my thesis.

To my family, I am profoundly grateful for the unending support, encouragement, and love you have provided me with over the years. Your unwavering belief in me has been a constant source of motivation.

I would also like to express my heartfelt thanks to Joana, for her unwavering support and understanding during the challenging times of this thesis. Your presence and encouragement were a constant source of strength. I would also like to extend my gratitude to Joana's family for their warmth and support.

Abstract

The rapid genetic characterization and surveillance of viral evolution are vital during outbreaks, as exemplified by recent occurrences such as the SARS-CoV-2 and Monkeypox outbreaks. Nonetheless, there exists an urgent need for comprehensive bioinformatics solutions to address this challenge. To meet this need, the user-friendly web-based platform known as INSaFLU has played a crucial role, enabling researchers with limited bioinformatics training to perform complex, multi-step analyses directly from raw sequencing data. However, INSaFLU has its limitations, such as the inability to operate on clusters and the complexity of its installation process. In response to these challenges, INSaFLU-Snakemake was developed. This consists on a reimplementation of INSaFLU with Snakemake, which not only replicates INSaFLU's functionality but also enhances its versatility, staying independent of the database and website. This implementation can be easily cloned and executed on local systems, SLURM clusters, and various other platforms. The ability to run on an SLURM cluster significantly reduces processing time, showcasing the benefits of parallelization and efficient resource allocation. Differences in output between INSaFLU and INSaFLU-Snakemake were observed, primarily attributed to variations in Python scripts and software versions. These differences were particularly noticeable in minor variant calling. The adoption of Snakemake as the Workflow Management System (WMS) opens up new possibilities for a more flexible and modular pipeline. To harness this potential, I introduced the option to choose between Snippy and iVar for variant calling and consensus generation within the pipeline. After comparisons with real and artificial samples, snippy and iVar showed similar results, but iVar has the advantage when resolving variants near primer regions. To test and use the new implementation of INSaFLU, 500 samples from the Portuguese Monkeypox dataset were analyzed with the ultimate goal of leveraging minor SNPs to enhance the correlation between samples and collection time. However, the incorporation of minor SNPs did not substantially improve the correlation. The dataset also revealed a significant proportion of clonal SNPs associated with APOBEC-mediated mutations agreeing with previous studies. In summary, INSaFLU-Snakemake is ready for immediate use and can easily be extended to meet evolving research needs.

Keywords: Snakemake, INSaFLU, SLURM, SARS-CoV-2, Monkeypox

Resumo

A rápida caraterização genética e a vigilância da evolução viral são vitais durante os surtos, como exemplificado por ocorrências recentes, como os surtos de SARS-CoV-2 e Monkeypox. No entanto, existe uma necessidade urgente de soluções bioinformáticas abrangentes para enfrentar este desafio. Para responder a esta necessidade, a plataforma de fácil utilização, conhecida como INSaFLU, tem desempenhado um papel crucial, permitindo que os investigadores com formação limitada em bioinformática efectuem análises complexas e em várias etapas diretamente a partir de dados brutos de sequenciação. No entanto, o INSaFLU tem as suas limitações, como a incapacidade de funcionar em clusters e a complexidade do seu processo de instalação. Em resposta a estes desafios, foi desenvolvido o INSaFLU-Snakemake, em que o pipeline INSaFLU foi implementado com o Snakemake, que não só replica a funcionalidade do INSaFLU como também aumenta a sua versatilidade, mantendo-se independente da base de dados e do website. Esta implementação pode ser facilmente clonada e executada em sistemas locais, clusters SLURM entre outras plataformas. A capacidade de execução num cluster SLURM reduz significativamente o tempo de processamento, mostrando os benefícios da paralelização e da alocação eficiente de recursos. Foram observadas diferenças nos resultados entre o INSaFLU e o INSaFLU-Snakemake, atribuídas principalmente a variações nos scripts Python e nas versões de software. Essas diferenças foram particularmente perceptíveis na identificação de variantes não clonais. A adoção do Snakemake como sistema de workflow management system (WMS) abre novas possibilidades para um pipeline mais flexível e modular. Para aproveitar este potencial, foi introduzida a opção de escolher entre o Snippy e o iVar para a identificação de variantes e a geração de consensos. Após comparações com amostras reais e artificiais, o Snippy e o iVar apresentaram resultados semelhantes, mas o iVar tem vantagem na resolução de variantes perto de regiões de primers. Para testar e utilizar a nova implementação do INSaFLU, foram analisadas 500 amostras do conjunto de dados do Monkeypox português, com o objetivo final de aproveitar SNPs não clonais para melhorar a correlação entre amostras e tempo de recolha. No entanto, a incorporação destes SNPs não melhorou substancialmente a correlação, sugerindo que outros factores podem

desempenhar um papel na evolução e divergência viral. O conjunto de dados também revelou uma proporção significativa de SNPs clonais associados a mutações mediadas por APOBEC, concordando com estudos anteriores. Em suma, o INSaFLU-Snakemake está pronto para utilização imediata e pode ser facilmente alargado para responder a novas necessidades de investigação.

Palavras-chave: Snakemake, INSaFLU, SLURM, SARS-CoV-2, Monkeypox

Contents

List of Figures

List of Tables

Acronyms

CWL Common Workflow Language *(pp. 6, 49)*

DAG Directed Acyclic Graph *(pp. 8, 24)*

HPC High-Performance Computing *(pp. 6, 17)*

iSNVs Intra-Host Single Nucleotide Variants *(pp. 3, 4)*

NGS Next Generation Sequencing *(pp. 1, 2)*

ONT Oxford Nanopore Technologies *(pp. 2, 3, 14, 24)*

SLURM Simple Linux Utility for Resource Management *(pp. 17–19, 48, 52)*

SNP Single Nucleotide Polymorphism *(pp. 3, 15, 17, 19, 20, 33, 36, 42, 44, 46, 47, 51, 53–55, 59)*

WMS Workflow Management Systems *(pp. 5, 6, 11)*

1 Introduction

The start of this decade showed the impact of viral surveillance with the SARS-CoV-2 pandemic and the Monkeypox outbreak [2–5]. Viral surveillance is a significant factor in public health, providing crucial insights to produce guidelines on how to act. Surveillance can be broken into three continuous steps: identifying individual cases, detecting patterns in the population, and finally, providing information to decision-makers about the population's health problems [6]. Next Generation Sequencing (NGS) was revolutionary in the genetic surveillance of viruses. Before this technology, most virus studies would sequence a known segment of the virus genome that would characterize a small percentage of the virus, making it more challenging to distinguish between closely related organisms. With NGS, it is possible to sequence the whole virus genome in a more time, material, and cost-efficient manner with the ability to detect rare variants. These variants are of great importance to point out the evolution of the virus[7, 8]. However, utilizing NGS in viral surveillance also brings new challenges in analyzing the vast sequencing data. Bioinformatics tools are essential to overcome these challenges. Each tool is specialized for specific steps in the analysis process, and using appropriate tools with the proper parameters is crucial to ensure reliable results. Automation is also vital in bioinformatics, as it enables the creation of pipelines that facilitate reproducibility and high-throughput data analysis. One example of NGS's power in viral surveillance is the INSaFLU project by the National Institute of Health Dr. Ricardo Jorge (INSA) in Portugal.

1.1 INSaFLU

INSaFLU [9] is a web-based suite that helps with viral surveillance allowing researchers to analyze whole-genome sequencing data easily. INSaFLU provides a highly tested, user-friendly platform that can be used by researchers with and without a background in bioinformatics to explore NGS data and generate required output data for real-time viral surveillance. Although initially developed for influenza, it now also supports SARS-CoV-2, Monkeypox, and others.

The bioinformatics pipeline developed and implemented in the INSaFLU web platform consists of six core steps: Read quality analysis and improvement; Type and sub-type identification; Coverage analysis; Variant detection and consensus generation; Alignment/phylogeny and Intra-host minor variant detection (and uncovering of putative mixed infections). INSaFLU has tools to analyze Illumina, with shorter reads and higher accuracy, and Oxford Nanopore Technologies (ONT), with ultra-long reads. The differences will be considered in each step, as necessary.

1.1.1 Read quality analysis and improvement

Whole genome sequencing analysis begins with evaluating the quality of "raw" reads (as they come from the sequencing machines) and improving/ensuring their quality. FastQC [10] is used to assess the quality of Illumina data, and Trimmomatic [11] is used for trimming. Similar quality control for ONT is done with NanoStat [12] and trimming with NanoFilt [12]. FastQC and Nanostat are quality control software that provides statistics about the reads. Trimmomatic and NanoFilt are trimming software that filters, and cuts reads based on specific parameters. This step results in the production of read quality reports and filtered reads that are ready to be used in subsequent analysis.

1.1.2 Type and sub-type identification

The step of type and subtype identification aims to provide the user with a quick estimate of the organism that is present in the sample. For this, it creates a draft de novo assembly from Illumina trimmed reads using SPAdes, screening the assembled contigs against an internal database of curated sequences using ABRicate [13]. In the case of ONT, it directly screens the ONT trimmed reads. The curated ABRicate database makes it possible to distinguish between multiple influenza types and subtypes, and other organisms such as SARS-CoV-2 and Monkeypox. SPAdes [14], a genome assembler, can assemble Illumina reads, and ABRicate is a tool that screens sequences against an internal database, returning details about possible matches. This step outputs the contigs obtained using the assembly (in the case of Illumina) as a fasta file and a table (as a tabulated file) of matches between the assembled contigs and the internal database.

1.1.3 Coverage Analysis

Coverage analysis [15] evaluates the mean depth of coverage when performing NGS. The analysis of coverage is performed by running a script on the depth file that returns the average depth of coverage for each locus, the proportion of the locus size covered by at least one read, and the proportion of the locus size covered by a user-defined value of reads. Color-coded outputs and depth of coverage plots for each locus per sample are also provided to help users identify atypical but relevant genetic events.

1.1.4 Variant detection and consensus generation

This step aims to identify Single Nucleotide Polymorphism (SNP)s and other genetic variations and create consensus nucleotide sequences from processed reads mapped against user-specified reference sequences.

Snippy [16] coordinates BWA [17] to map the reads against the user-defined reference for Illumina reads. It utilizes Freebayes [18] for variant calling, and SnpEff [19] for variant annotation. BCFtools is utilized to generate the consensus sequences. Snippy manages all the software and data in this step.

The script Medaka_consensus [20] maps the reads for ONT reads with minimap2 [21], and variants are called with the Medaka_variant script. Annotation of the variants (a prediction of its potential effects) is also performed using SnpEff. The consensus sequences are generated using BCFtools [22].

Low-coverage regions are masked from the consensus based on a user-defined threshold. The user can mask the consensus in specific nucleotides or ranges, replacing them with Ns. The resulting consensus sequences provide a foundation for making phylogenetic inferences and tracking the evolution of the virus.

This step outputs an annotated variant file (as a tabulated file) and the consensus of each sample (as a fasta file).

1.1.5 Alignment/Phylogeny

After every sample has generated consensus and passed a filter by horizontal coverage above the user-defined threshold, the consensus sequences are ready for further analysis. INSaFLU uses MAFFT [23] to align consensus nucleotide sequences. It also generates amino acid alignments and phylogenetic trees using MAFFT and FastTree2 [24], respectively. The alignments and trees are updated as more data becomes available, and they can be downloaded or explored using MSAViewer [25] and PhyloCanvas [26] on the website.

1.1.6 Intra-host minor variant detection (and uncovering of putative mixed infections)

INSaFLU enables the analysis of intra-host single nucleotide variants Intra-Host Single Nucleotide Variants (iSNVs) in virus samples. The sample's mapping files are subjected to the freebayes software, which applies criteria such as minimum mapping and base quality, minimum depth of coverage, and the number of reads supporting an alternative allele. As more samples are added to the INSaFLU project, a table is updated to include iSNVs with a frequency of less than 50%. Additionally, a plot of the percentage of iSNVs in the frequency ranges of 1–50% and 50–90% is included to help identify mixed infections, which occur when a patient is infected with viruses from two different genetic backgrounds or can be a sign of contamination.

INSaFLU flags samples as putative mixed infections based on specific criteria involving the number of iSNVs in these frequency ranges and their sum. These criteria were determined through testing with replicates of "true" mixed infections and with artificial mixtures of viruses.

1.1.7 Issues

INSaFLU was developed specifically to bridge the gap between bioinformatics experts and wet lab investigators. Traditionally, most data analysis pipelines in this field are executed through the terminal, which can be a barrier for researchers who are more focused on wet lab experiments and lack extensive training in bioinformatics. To address this issue, INSaFLU was created as a suite of web-based tools that offer a user-friendly interface for analyzing and exploring genomic data.

One of the key advantages of INSaFLU is its comprehensive and well-documented user interface, which allows researchers to interact with the data analysis process without needing to delve into the intricacies of command-line tools. However, a challenge arises during the installation and setup of the software on local systems for users who need to retain data in-house or establish a dedicated processing queue. The installation process can be intricate and potentially intimidating for users with limited technical expertise. To address this issue, an alternative installation method using Docker is provided. Docker simplifies the installation by reducing the number of steps and configuration requirements. This approach aims to make both the installation and usage of INSaFLU more accessible.

While the Docker implementation enhances accessibility, it introduces a trade-off in terms of potential differences between the regular INSaFLU version and the INSaFLU-docker version. These differences could potentially lead to challenges in reproducing results consistently, which is crucial for scientific research. Therefore, while Docker simplifies the installation, users need to be aware of these potential discrepancies.

Despite its benefits, INSaFLU and its Docker implementation have limitations. For researchers who intend to run the data analysis pipeline in alternative environments, such as high-performance computing clusters or cloud platforms, the platform's adaptability is restricted. Additionally, if a user desires to modify only a portion of the pipeline, the website is inflexible even though the code is publicly available.

1.2 Tools to help follow FAIR principles

The FAIR principles[27] are a set of guidelines for making scientific data more Findable, Accessible, Interoperable, and Reusable by both humans and machines. They aim to improve the infrastructure supporting the reuse of data and enhance the ability of machines to automatically find and use it. The FAIR principles, which can be applied to any type of

digital object, such as data, algorithms, tools, or workflows, play a crucial role in addressing one of the main issues with scientific data: the challenge of reproducing results from data analysis. The following sections describe tools that can help improve reproducibility.

1.2.1 Conda

A crucial aspect of addressing reproducibility concerns is the assurance of using identical software versions. To tackle this challenge, tools like Conda [28] emerge as valuable solutions. Serving as a package and environment manager compatible with Windows, macOS, and Linux, Conda not only accommodates programming languages like Python but also accelerates tasks such as swift dependency installation, maintenance, and environment setup, ensuring smooth transitions between different setups. An advantage offered by Conda is the capability to establish distinct environments for running various versions of identical software, meanwhile maintaining the integrity of the environment manager unchanged.

1.2.2 Docker

Docker [29] is an open-source containerization platform that simplifies the packaging and deployment of lightweight and portable applications and services consistently and in a reproducible way.

Docker containers contain all the dependencies, libraries, and code necessary to run an application and can operate reliably across various environments and platforms. It is especially suited for complex applications because they can be broken down into smaller, independent components that can be developed, tested, and deployed separately.

1.2.3 Workflow Management systems

Workflow Management Systems (WMS) automate computational analyses by combining individual, discrete data processing tasks into cohesive pipelines. They provide an interface that simplifies the movement and processing of data and manages dependencies between tasks [30]. Regardless of specific WMSs, adopting such systems helps adhere to the FAIR principles.

WMSs can be categorized based on the tools used to build them [30, 31]. For example, some systems have a graphical user interface that makes learning and using them more accessible, like galaxy[32]. Other systems, such as Snakemake[33] use libraries associated with programming languages such as Python. Some use a domain-specific language that provides specialized statements and declarations to model central workflow management components, such as Nextflow [19]. Meanwhile, system-independent workflow specification languages also have a declarative syntax that can be parsed and executed by any executor that implements them. Examples of system-independent workflow specification

languages include the Common Workflow Language (CWL), the Workflow Description Language, and Nextflow.

The CWL originated in 2014 and achieved its first stable release (CWL v1.0) in 2016. It is an open standard that describes analysis workflows and tools, driven by the need for scalable and portable workflows across different environments [34]. CWL is classified as a WMS specifically designed to orchestrate and execute complex data processing pipelines. Workflow specifications in CWL are written using the YAML or JSON format, utilizing a declarative syntax to define workflows and tools, including inputs, outputs, and sequential data analysis steps. Execution engines, such as cwltool, Toil, and Arvados, ensure reusability and reproducibility of data analysis pipelines across various platforms, from local machines to high-performance clusters and cloud environments [35]. A notable feature of CWL is its platform-agnostic design, enabling its implementation on diverse platforms. This compatibility is further enhanced by seamless integration with Docker and Conda, which encapsulate and distribute software tools and dependencies, ensuring workflow portability and reproducibility. CWL is a versatile and efficient WMS that suits various applications and environments, offering robustness, flexibility, and language independence.

Although CWL has numerous qualities as WMS, the evolution of Snakemake has led it to offer many of the same advantages. Its unique integration with Python continues to differentiate it, rendering the chosen WMS for this project, due to its ease of configuration and adaptability for extension.

1.2.4 Snakemake

Snakemake [36] is a WMS that occupies a middle ground between GUI-based and fully declarative WMS. It is built on top of Python, which is advantageous due to its popularity. Snakemake operates on the same principles as makefiles, generating a chain of rules with associated actions to produce intermediate outputs until the final desired output is obtained. It provides advanced features like parallel execution, Conda, and Docker integration, making it ideal for data analysis in bioinformatics and computational biology, ensuring reproducibility and scalability.

The principles on which Snakemake was built align with the FAIR principles for building workflows. The primary concept of trusting scientific results is reproducibility, automation, scalability, portability, and accessibility to the results, as well as the code and parameters used to generate those results. Furthermore, if the workflow comprises reusable parts that can be used in other projects, it respects the last principle, reusability.

It is a versatile WMS that seamlessly runs on different platforms, including Linux, macOS, and Windows. It is also compatible with High-Performance Computing (High-Performance Computing (HPC)) clusters and cloud services like AWS.

1.2.4.1 Rules

```
rule example_rule:
    input:
        "samples/{sample}/previous_tool/{sample}.file",
    output:
        "samples/{sample}/tool/output.fasta",
        dir=directory("samples/{sample}/tool/"),
    conda:
        "../envs/example.yaml"
    threads: config["example_threads"]
    params:
        "--specific-param",
    resources:
        mem_mb=memory["tool"],
    log:
        "logs/samples/{sample}/tool.log",
    benchmark:
        "benchmark/samples/{sample}/tool.tsv"
    shell:
        "tool -t {threads} {params} -s {input} -o {output.dir}"
```

A Snakemake rule is a type of directive that specifies how an output file should be generated. Each rule defines the inputs and outputs required for a particular step in a pipeline, as well as any parameters or options necessary for the execution of the command. One of the defining features of Snakemake is using placeholders, such as "{sample}", to represent dynamic values determined at runtime. During the execution of the workflow, these placeholders are replaced by specific values, typically provided in a configuration file. The configuration files specify the sample names, allowing Snakemake to generate the correct input and output file paths.

The "example_rule" rule, Snakemake file takes a single input file, "{sample}.file", located in the "samples/{sample}/previous_tool/" directory. The expected output of the rule is a single file, "output.fasta", located in the "samples/{sample}/tool/" directory. During the execution of the workflow, Snakemake replaces "{sample}" with the appropriate values from the configuration files, resulting in the correct input file path for each sample.

Snakemake keywords used in the rule include "input" and "output" for defining the input and output files, "directory" for creating a directory for the output (optional), and "conda" for specifying the conda environment. The "params" keyword allows optional parameters to be passed to the shell command, while "resources", such as the memory and time limits, specify the necessary resources. The "log" and "benchmark" record information about the execution and threads specifies the number of threads. These keywords ensure consistent and reproducible execution of each rule, essential in complex bioinformatics workflows.

1.2.4.2 The Workflow Logical Structure as a Directed Acyclic Graph

In the context of Snakemake the Directed Acyclic Graph (DAG) represents the workflow's logical structure, showing the relationships between rules, input, and output files. Each rule is associated with its corresponding input and output files. Snakemake generates a DAG by parsing the rules specified in the Snakemake files. The DAG plays a crucial role in guiding Snakemake's execution process. Snakemake checks the modification times of input and output files when executing the workflow. It will only run a rule if input files are newer than output files or if any dependencies have changed. This ensures that the workflow remains up-to-date and efficient. Visualizing the DAG gives users valuable insights into the data flow and dependencies between input and output files. This visibility facilitates a streamlined and efficient analysis pipeline, enabling better control over the workflow and enhancing reproducibility.

1.2.4.3 Rulegraph

The rulegraph is a graph-based representation of the dependencies between the different rules in a Snakemake workflow. In a rulegraph, each rule is represented as a node, and the dependencies between the rules are represented as edges. The direction of the edges indicates the order in which rules need to be executed to complete the workflow successfully. The rulegraph is generated automatically by Snakemake based on the rules defined in the workflow script. The rule graph retains a constant structure regardless of the number of samples processed.

1.2.4.4 Checkpoints

Checkpoints in Snakemake provide a powerful mechanism for dynamic changes and additions to the workflow during runtime, enhancing the flexibility and efficiency of computational workflows. By completing a checkpoint, new files are appended to the expected output list, and their rules are seamlessly integrated into the workflow's DAG. This integration allows for re-evaluating rules within the DAG and adding new jobs as necessary. Consequently, checkpoints enable dynamic workflow modification, making the pipeline more adaptable.

1.2.4.5 Benchmark

The "benchmark" keyword in the rule definition enables the retrieval of memory and CPU time information associated with each rule's execution. This information helps optimize workflows, identify potential bottlenecks, and estimate resource requirements for future runs. By using the benchmark keyword in Snakemake rules, users can gain valuable insights into the performance of their workflows and make informed decisions about resource allocation.

2

Aims

This work aims to develop a pipeline in a workflow management system that can reproduce the pipeline used in the INSaFLU website, independent of a database and graphical interface, able to run in various environments, such as personal computers and clusters. Snakemake was chosen as the WMS to achieve a system that allows for the modification and addition of pipeline modules to test alternative tools for analyzing viral genomes. As a case study, this thesis will use the INSaFLU-Snakemake pipeline and SLURM to explore the Portuguese Monkeypox data.

2.1 INSaFLU-Snakemake

The primary objective is to replicate the reference-based genomic surveillance pipeline available in INSaFLU. The goal is to achieve the same output with both pipelines starting with the same samples and parameters.

2.2 Extending INSaFLU-Snakemake

One of the aims of developing this WMS replicating INSaFLU is to have a system that enables quick testing of new software/pipelines in INSaFLU. For this thesis, we will demonstrate the adaptability of the developed WMS by developing an alternative module to snippy that is iVar. The goal is to perform extensive testing and conduct result comparisons to assess the performance and accuracy of snippy and iVar in viral data analysis. The workflow management system will integrate various tools within the analysis pipeline. By leveraging this system, the two tools can be comprehensively evaluated, leading to a deeper understanding of their strengths and limitations in handling viral data. The similarities will be verified by comparing 1110 samples of SARS-CoV-2 and the accuracy of each tool will be tested using artificial samples.

2.3 Enabling execution in a SLURM cluster

INSaFLU is designed to be user-friendly and accessible for anyone who wants to analyze viral data. However, running the pipeline in a cluster environment may be more efficient and convenient for large-scale datasets. This work will demonstrate how to run INSaFLU-Snakemake in a cluster and the main advantages and challenges of this approach.

2.4 Showcase INSaFLU-Snakemake by analyzing the Portuguese Monkeypox dataset

The recent Monkeypox outbreak in Europe led to Portugal's significant sequencing effort of sequencing around 500 samples, half of the diagnosed cases in the country. The Monkeypox dataset will be used to showcase the application of INSaFLU-Snakemake and its extension with iVar, exploring some of its results. Notably, the Monkeypox virus is known to have few mutations, which presents a challenge in resolving explosive outbreaks like this one. Therefore, the objective is to investigate whether minor variants could address this challenge and provide more context to the phylogenetic and epidemiological patterns among the 500 samples, achieving a higher resolution in Monkeypox analysis.

3

Methods

3.1 INSaFLU-Snakemake

3.1.1 Description of INSaFLU-Snakemake

I decided to use Snakemake as the WMS for this project as it is closely intertwined with Python, seamlessly integrates with Conda, and facilitates execution in diverse environments such as clusters and the cloud. INSaFLU-Snakemake has 2 execution modes. The ”sample analysis” in which it only runs the ”Read quality analysis and improvement” and optionally ”Type and sub-type identification”. The ”project” mode runs all the modules present in the pipeline including the two modules in ”sample analysis” mode.

3.1.2 Minimum Dependencies

INSaFLU-Snakemake has a few software dependencies (Tab. 3.1) that can be installed with conda. After installing them, Snakemake will take care of the remaining dependencies needed to execute the pipeline, also with conda.

Table 3.1: INSaFLU-Snakemake dependencies.

INSaFLU-Snakemake Dependencies
snakemake-minimal =7.19.1
python =3.10
biopython =1.79
pyyaml =6.0
pandas =1.4.3

3.1.3 Project Configuration and Structure

The project configuration follows the standard structure for Snakemake Workflows (Fig. 3.1). The configuration files are situated in two distinct directories: the ”config” folder and the ”user” folder. The ”config” folder has information that is independent of sample types and project configurations. The ”constants.yaml” file is in the ”config” folder and serves as a reference point for all other required files and directories essential for executing

the pipeline. Additionally, the ”threads.yaml” file is present in this directory, defining the number of threads allocated for each pipeline rule that can take advantage of them. The file ”max_memory.yaml” is also contained in this folder, determining the maximum memory allocation in megabytes for each rule within the pipeline.

On the other hand, the ”user” directory is where configuration files that define each run are stored. The ”project.yaml” file defines the run's configuration, including the selection of references, the project name, and the run type. Descriptions of sample names, files, and the technology used to generate them are outlined within the ”samples.csv” file. The entries for fastq1 and fastq2 should exclude the fastq.gz extension and sample names must be unique. The ”parameters.yaml” file encompasses all customizable parameters within the pipeline software. The ”references” directory houses the fasta and genbank files. The ”primers” folder contains fasta files along with their corresponding pair_information.tsv files. The fastq.gz files are in the ”reads” folder.

The ”workflow” directory serves as the central hub for defining the workflow's logic and dependencies. The main entry point for INSaFLU-Snakemake is the ”Snakefile”. In this file, preprocessing occurs to gather the necessary information for defining the rules to be executed. Dependencies for each rule are stored in the ”envs” directory using yaml files. Rules, which are defined in smk files specific to the Snakemake, are located in the ”rules” folder. These files extend the capabilities of Python by incorporating additional keywords. The ”scripts” folder contains Python scripts that implement the logic for various pipeline steps. The ”tests” directory holds test files for the Python scripts to ensure their functionality. Lastly, the ”db” directory stores information related to abricate and snpeff.

The ”slurm” folder handles the execution of tasks through slurm. The ”config.yaml” file within this folder defines key-value pairs that specify the resources to be utilized. The ”run_slurm.sh” is the file that is submitted to sbatch to initiate the pipeline execution.

The results of the pipeline are located within the results directory. This folder will only be visible after the first pipeline execution and it has the following structure:

```
1 results (directory)
2     samples
3     align_samples
4     projects
```

Within the ”results/samples” directory, separate directories for each sample containing FastQC results and trimmed reads can be found. The ”results/align_samples” directory encompasses the outputs of variant calling and consensus assembly. Project-related files are stored within the ”results/projects” directory. The primary files reside in the ”results/projects/project_name/main_result” folder, which encompasses alignments, variants, and coverage files.

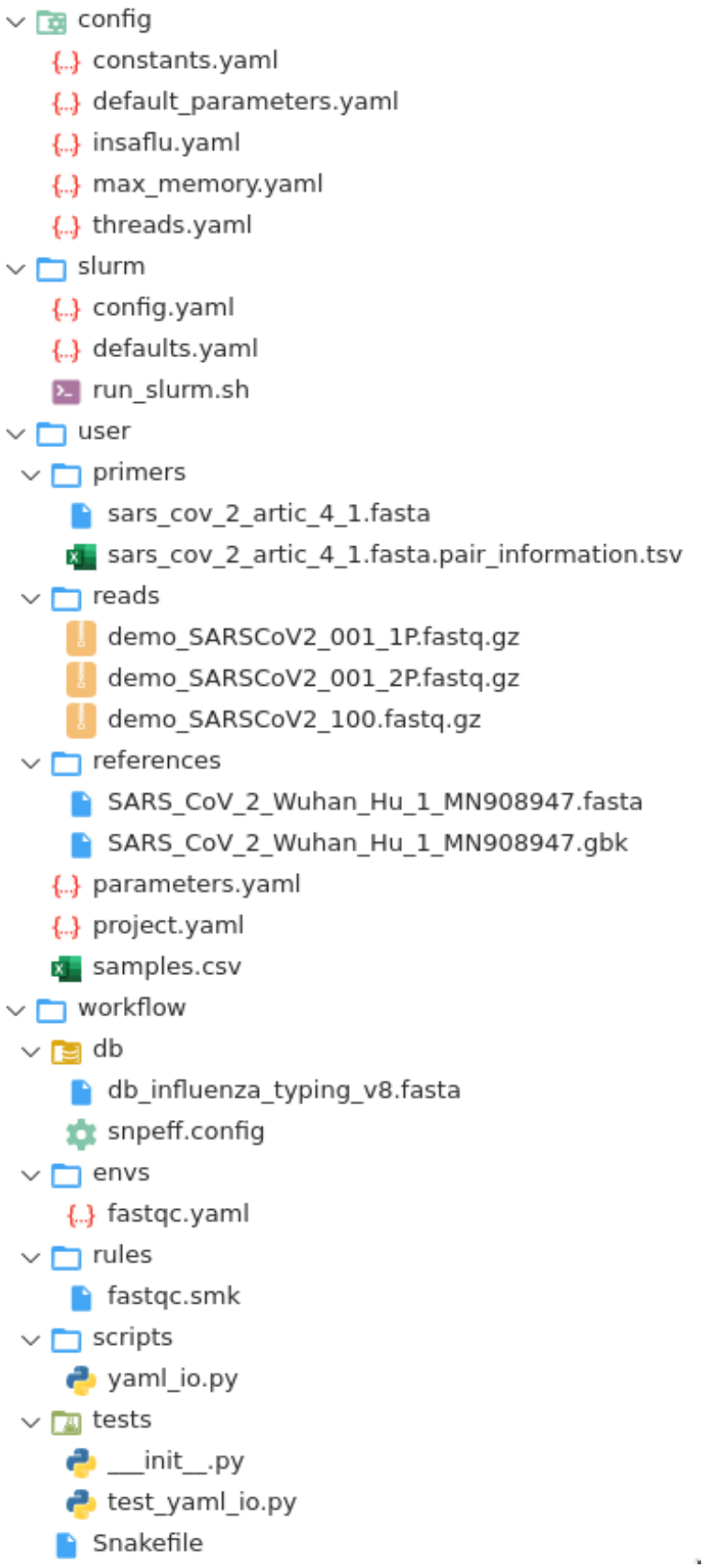

Figure 3.1: INSaFLU-Snakemake project structure.

3.1.4 Implementing the functionality

3.1.4.1 Rule structure

I wrote rules based on software or entire modules, although when the modules were too complex, they were separated into different rules or files. All the rules are imported in the Snakefile.

3.1.4.2 Checkpoints

In "project" mode, the pipeline interacts with checkpoints. These checkpoints serve a crucial purpose, confirming whether the reference is SARS-CoV-2 and whether any of the samples exhibit horizontal coverage beyond a user-defined threshold during execution.

The checkpoint creation and utilization process involves two key aspects. Firstly, a checkpoint is defined using the keyword "checkpoint" in place of "rule." Secondly, a rule, class, or function is designed to wait their execution until the corresponding checkpoint is successfully passed, allowing the workflow to continue or end its execution.

In the context of INSaFLU-Snakemake, this mechanism is realized through the use of classes. These classes are invoked through the __call__ method, which is specifically defined. These classes operate on the output of the checkpoints and return a list of output files if certain conditions are met, otherwise, an empty list is returned. The return value is appended to the list of expected output files.

3.1.5 Comparing Snakemake vs. Website

To test whether the INSaFLU-Snakemake was working as expected, I used a diversity of samples to compare the results between the public website and the Snakemake implementation. The samples used in the comparison were 20 Illumina and 20 ONT from SARS-CoV-2, 21 Illumina from Monkeypox, and 13 Illumina from Flu. Monkeypox can be used to test the pipeline on larger genomes and the Flu to test how it handles multi-segmented genomes. The most effective means of validating the Snakemake replica's success is to use the same input parameters and examine the output files. Any discrepancy between the files indicates that the INSaFLU-Snakemake is performing a distinct operation. The files selected for this comparative analysis include the consensus, alignment, amino acid alignment, validated variants, minor validated variants, and the coverage file. The comparison of the consensus, alignment, and amino acid alignment requires the evaluation of each nucleotide of every sample. It is expected to have the same samples present in each file and no difference between them. The validated and minor validated variants will be compared based on the sample, locus, reference nucleotide, altered nucleotide, and frequency. Lastly, the coverage analysis will verify the differences in the genome coverage.

3.2 Extending INSaFLU-Snakemake

The first step in creating a modular pipeline is to allow users to choose the software. I achieve this by implementing a dynamic variable related to the folder where the expected processed files will be (project.yaml). This variable can change the pipeline's output depending on the chosen software. The second step is to connect the chosen software with all the previous and following rules. This ensures the pipeline remains consistent and coherent, regardless of the software choice. To test this, I implemented an extension of the pipeline by using iVar as an alternative to snippy.

3.2.1 Comparing alternative pipeline outputs

Tests were conducted using both real and artificial data to assess similarities and differences between the different alternatives, as well as their performance and accuracy. This allows a better assessment of the similarities and differences between the software.

3.2.1.1 Using real samples

To compare snippy and iVar, I used 1110 samples from late December 2022 and January 2023 of the national SARS-CoV-2 genetic surveillance system. The pipeline with the parameters in the table 3.2 used Snippy and iVar to generate consensus sequences, which were then aligned. Each nucleotide position was compared and classified based on its comparison with the other tool's consensus sequence. Positions with matching nucleotides were labeled as Matches. Positions with mismatching nucleotides were labeled as Mismatches. Those positions where Snippy's consensus sequence contained an N but iVar's consensus sequence had a nucleotide were labeled Snippy Conservative. Similarly, positions where iVar's consensus sequence contained an N but Snippy's consensus sequence with a nucleotide was labeled iVar Conservative. Positions with an N in Snippy and a gap in iVar were labeled gapSnippyNiVar. When the opposite occurred, they were labeled NSnippygapiVar. Finally, positions where iVar's consensus sequence had a nucleotide that differed from "ATGC" were labeled as uncovered cases.

3.2.1.2 Using artificial samples

To rigorously assess the accuracy of two software tools, I created a methodology employing both real and artificially generated data. My focus was on analyzing the genomic variations of 44,625 Portuguese SARS-CoV-2 samples collected between 2019 and 2023. Through a linear regression analysis, I observed an average mutation rate of approximately 30 SNPs per year (see Fig. .3). Extrapolating this rate, I predict that by 2040, the SARS-CoV-2 genome could accumulate up to 500 SNPs. Notably, these mutations appeared to concentrate prominently in the S protein coding region (Fig. .4), which by 2022 exhibited the highest SNP frequency compared to any other genomic regions (Figs. .1 and .2).

Table 3.2: Parameters used to analyze real samples with snippy and iVar.

Key	**Value**
min_coverage_consensus	90
Trimmomatic	
trimmo-ILLUMINACLIP	null
trimmo-HEADCROP	null
trimmo-CROP	null
trimmo-SLIDINGWINDOW1	5
trimmo-SLIDINGWINDOW2	20
trimmo-LEADING	3
trimmo-TRAILING	3
trimmo-MINLEN	35
Nanofilt	
QUALITY	8
LENGTH	50
HEADCROP	30
TAI5ROP	30
MAXLENGTH	50000
Snippy/iVar	
mapqual	20
mincov	30
minfrac	0.51
Medaka	
Medaka_model	r941_min_high_g360
mincov_medaka	30
min_prop_for_variant_evidence	0.80
Mask Positions	
MASK_SITES	null
MASK_REGIONS	null
MASK_F_BEGINNING	null
MASK_F_END	null

To mimic the natural evolution of the SARS-CoV-2 genome, I devised a unique approach using artificial data. I used the frequency of mutations at each position as the probability for incorporating the mutation into the new artificial samples. By iteratively adjusting the probabilities for each position, I generated artificial genomes with the anticipated number of mutations. These artificial genomes were then saved as fasta files. Subsequently, I used the ART software [37], configuring it with relevant parameters, to create fastq files simulating Illumina sequencing data, based on mean fragment lengths and read lengths collected from SARS-CoV-2 samples. Our evaluation involved a comparative analysis between Snippy and iVar, both of which underwent parallel processes from artificial sample generation to consensus sequence analysis.

To easily test the new tool, a new pipeline was created with the INSaFLU-Snakemake modules that were reused and connected to the generation of the artificial reads and the

consensus analysis. This approach allowed for the identification of discrepancies between sample and reference sequences, streamlining genomic data analysis. The code for the pipeline is in the GitHub repository RdrigoE/compare_assembler_consensus.

When comparing the outputs of Snippy and iVar from real sequencing data, it is only possible to infer the dissimilarities between them. To address this, I introduced artificial samples, systematically increasing SNP counts up to 1000 (0, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000), and each condition with 100 samples. This assessment measured the software's ability to manage expanding sequence diversity using the current reference genome. Moreover, I explored the performance of both tools with artificially reduced identity percentages in the spike gene (100,95,90,85,80,75,70,50) reflecting the presence of multiple SNPs in a concentrated area, once again, each condition with 100 samples. This study centers around software accuracy, the software's ability to detect SNPs, mismatches that occur when the software and artificial reference possess distinct nucleotides, and the quantification of Ns within the consensus sequence. This workflow presents a comprehensive methodology for quantitatively evaluating tool performance under varying genomic conditions.

3.2.1.3 Edge Cases

The performance comparison of three variant calling methods—snippy, snippy with trimmomatic primers, and iVar—was conducted using specific sets of samples that were known *apriori* to contain problematic *loci*. In the case of SARS-CoV-2 analysis, five samples were used to investigate a challenging mutation at position 22789. This mutation involved an SNP from A to C, which proved problematic for the snippy tool. The goal was to assess the ability of each method to accurately detect this mutation. For the Monkeypox analysis, three positions were examined. Notably, at position 13068, snippy exhibited false positive SNP calls in three samples. Similarly, at position 13090, erroneous SNP calls were identified by snippy in six samples. Lastly, at position 12836, the first sample should not call the SNP, while in another sample, a minor SNP that should be clonal was detected. This comprehensive evaluation provided insights into the performance characteristics of the variant calling tools under different genomic scenarios.

3.3 Enabling execution in a SLURM cluster

3.3.1 Configuring Snakemake to run SLURM

HPC clusters often become crucial for operating these pipelines when dealing with large datasets. Simple Linux Utility for Resource Management (SLURM), a well-recognized cluster management system, facilitates users to send jobs to the cluster and supervise their progression. To execute the INSaFLU-Snakemake workflow on an SLURM cluster, users need to configure SLURM settings either through a comprehensive command line

Table 3.3: Example SLURM "config.yaml" file.

Key	Value
partition	<PartitionX>
qos	<QueueX>
mem_mb	1000
restart-times	3
max-jobs-per-second	10
max-status-checks-per-second	1
local-cores	1
latency-wait	60
jobs	500
keep-going	TRUE
rerun-incomplete	TRUE
printshellcmds	TRUE
scheduler	greedy
use-conda	TRUE

approach or by utilizing a configuration file, followed by the invocation of a corresponding configuration profile. A YAML file serves as the blueprint for defining the execution configuration tailored to SLURM's requirements. This configuration encompasses vital parameters such as the maximum allowable runtime for jobs, memory prerequisites, and CPU count, as well as the designated partition and queue. Further insights are available in Table 3.3.

For a more explicit invocation:

```
snakemake --slurm --default-resources
slurm_account=<your SLURM account>
slurm_partition=<your SLURM partition>
```

Alternatively, a configuration profile can be employed:

```
snakemake --profile slurm
```

These streamlined approaches empower users to effectively harness the potential of SLURM-based clusters, ensuring efficient management of resource-intensive computational tasks.

3.3.2 Benchmarking

Choosing between running tasks on a single computer or a SLURM cluster is important in high-performance computing. Benchmarking, a process of carefully evaluating and comparing performances, serves as a tool to determine which approach is more efficient. By analyzing factors such as the time it takes for tasks to complete and the amount of resources they consume, benchmarking provides valuable insights that guide how resources are allocated and how well the system can grow. This analysis facilitates informed decision-making to address the increasing computational requirements of applications

effectively. I used 50 samples in this study from the Portuguese Monkeypox Dataset. In order to assess the pipeline's performance, a comparative study of its execution on a single node versus on an SLURM cluster was conducted. The evaluation mainly focuses on two critical metrics: execution time and resource utilization. Execution time will be clocked from when Snakemake is invoked until completion. Resource utilization will be examined by scrutinizing the CPU time, the time taken to complete the rule and memory consumption. This evaluation will help better understand the efficiency and scalability of the pipeline under different computational environments.

3.4 Showcase INSaFLU-Snakemake by analyzing the Portuguese Monkeypox dataset

3.4.1 Sample Description

To showcase INSaFLU-Snakemake I used 500 Monkeypox samples from the 2022 outbreak in Portugal. The samples have two origins, 49 are from metagenomic sequencing, and the remaining 451 are from 4 different amplicons [38] batches. All samples were sequenced using Illumina.

3.4.2 Parameters used for the analysis

The Monkeypox research used standard quality control and Illumina data processing settings. Notably, the threshold for read selection was adjusted to 30 to enhance confidence in clonal variants and the minimum sample coverage was raised to 90, the parameter adjustments aimed to guarantee all samples' reliability and associated variants used in the bioinformatic analysis. The monkeypox_MT903344_MPXV_UK_P2_wo_NNcut_191315 reference is the Monkeypox genome from 2022 with a cutoff at the position 191315 to remove one of the two identical terminals, having less 5919 nucleotides than the original one. The primers were the MPXV MT903345 Yale PrimalSeq [39].

3.4.3 Quality Control of Results

To assess the quality of the results, I used information from the literature. Namely, information from Isidro et al.[40], where they identified 46 SNPs present in most of the samples and exhibited a strong mutation bias. These included 24 non-synonymous, 18 synonymous, and four intergenic nucleotide replacements with the APOBEC pattern GA > AA and TC > TT. This study will evaluate and discuss the presence of these 41 SNPs due to 5 of them being after the cutoff in the reference used.

3.4.4 Acessing the impact of using minor variants to increase granularity

To see how informative would minor variants be to distinguish the different samples, I integrated minor variants into the sample's consensus sequences. This strategy aims

to amplify identifiable differences between samples, thus widening their phylogenetic distance and contributing to a more complex structural representation than is currently available. To assess the effectiveness of this approach, I calculated the correlation between the collection time of each sample and its respective distance in the phylogenetic tree. Should the introduction of minor SNPs improve this correlation, it would indicate their value in explaining the relationships among the samples, thereby proving their inclusion in the analytical process.

The correlation between genetic distance and time of sample collection was assessed using the Pearson product-moment correlation coefficients. This statistical measure describes the strength and direction of a linear relationship between two continuous variables. The Python library, numpy[41], was utilized for this computation. The genetic distance between the different Monkeypox samples was computed using the Hamilton distance metric, commonly used in genomics to quantify the difference between two sequences. This computation was performed using the Reportree[42] software, a flexible pipeline that facilitates the detection of genetic clusters and their linkage to epidemiological data. The Hamilton distance offers a straightforward means of quantifying the genetic disparity between samples by adding the number of loci at which the two sequences differ.

4

RESULTS

4.1 INSaFLU-Snakemake

The Snakemake-based version of the INSaFLU consists of 87 structured rules and two checkpoints in 21 files. The pipeline's source code is hosted on GitHub at www.github.com/RdrigoE/insaflu_snakemake. The pipeline utilizes Git and GitHub actions for continuous evaluation to ensure quality and consistency, and this facilitates the early detection of potential anomalies with each code update. This pipeline was only tested on Ubuntu and WSL-Ubuntu.

4.1.1 Usage

4.1.1.1 Installation Process

Download and install Mambaforge.

```
curl -L link/to/Mambaforge-Linux-x86_64.sh -o Mambaforge-Linux-x86_64.sh
bash Mambaforge-Linux-x86_64.sh
```

Or if conda is already installed, it possible to install mamba in the base environment:

```
conda activate
conda install -c conda-forge mamba
```

Clone the INSaFLU-Snakemake git repository.

```
git clone https://github.com/RdrigoE/insaflu_snakemake.git
cd insaflu_snakemake
```

4.1.1.2 Activate Environment

Activate the conda base environment, and create a new environment to execute the Snakemake.

```
conda activate base
mamba env create --name insaflu --file config/insaflu.yaml
conda activate insaflu
```

4.1.1.3 Example Configuration

The presence of fastq files can be verified by the user in the user/reads directory:

```
sample_001_1P.fastq.gz
sample_001_2P.fastq.gz
sample_002.fastq.gz
sample_003.fastq.gz
```

The metadata on the samples to process is in user/samples.csv (Tab. 4.1). The sample_001 is a pair-ended Illumina sample, the sample_002 is a single-end Illumina sample and sample_003 is an ont sample. All of these samples are from SARS-CoV-2 genomes.

The project configuration is in the user/project.yaml (Tab. 4.3). To run just the sample analysis the "only_samples" should be set to true, but in this case, it is false so it runs as a project, therefore, it will run the sample preprocessing and all the other steps. The classification of the samples is set to true in the "abricate" parameter. The project name is set in the "project_name" key with the value of "sars_cov_2_2020" as the samples are from 2020. The fasta and genbank references are the Wuhan sample from 2019 [43]. The Illumina software to get the variants and consensus is set in the "Illumina_consensus" key and the values can be snippy or iVar, the option being snippy, there is no need to specify a primer.

The example of "parameters.yaml" file is structured by key-value pairs as the table 4.2 shows.

Table 4.1: Example of "samples.csv" file to describe the samples used in INSaFLU-Snakemake.

sample_name	fastq1	fastq2	tech
sample_001	sample_01_001	sample_01_002	Illumina
sample_002	sample_002		Illumina
sample_003	sample_003		ont

4.1.1.4 Execution

To locally execute the workflow, the following command should be run:

```
snakemake -c 8 --use-conda
```

The command-line option "-c" represents "–cores," indicating the desired thread count for Snakemake's management. The flag, "–use-conda," serves to leverage the conda environments defined within each rule. Without this specification, the software should be accessible via the system's $PATH.

4.1.2 Checkpoints

The pipeline has two checkpoints, one just for SARS-CoV-2 and another one that is computed in every project run. The checkpoint dedicated to SARS-CoV-2 uses the Abricate

Table 4.2: Example of ”parameters.yaml” file in INSaFLU-Snakemake.

Key	**Value**
min_coverage_consensus	70
Trimmomatic	
trimmo-ILLUMINACLIP	null
trimmo-HEADCROP	null
trimmo-CROP	null
trimmo-SLIDINGWINDOW1	5
trimmo-SLIDINGWINDOW2	20
trimmo-LEADING	3
trimmo-TRAILING	3
trimmo-MINLEN	35
Nanofilt	
QUALITY	8
LENGTH	50
HEADCROP	30
TAILCROP	30
MAXLENGTH	50000
Snippy/iVar	
mapqual	20
mincov	10
minfrac	0.51
Medaka	
Medaka_model	r941_min_high_g360
mincov_medaka	30
min_prop_for_variant_evidence	0.80
Mask Positions	
MASK_SITES	null
MASK_REGIONS	null
MASK_F_BEGINNING	null
MASK_F_END	null

Table 4.3: Example of ”project.yaml” file in INSaFLU-Snakemake.

KEY	**VALUE**
only_samples	False
abricate	True
project_name	sars_cov_2_2020
fasta_reference	SARS_CoV_2_COVID_19_Wuhan_Hu_1_MN908947.fasta
gb_reference	SARS_CoV_2_COVID_19_Wuhan_Hu_1_MN908947.gbk
Illumina_consensus	snippy
primers_fasta	Null

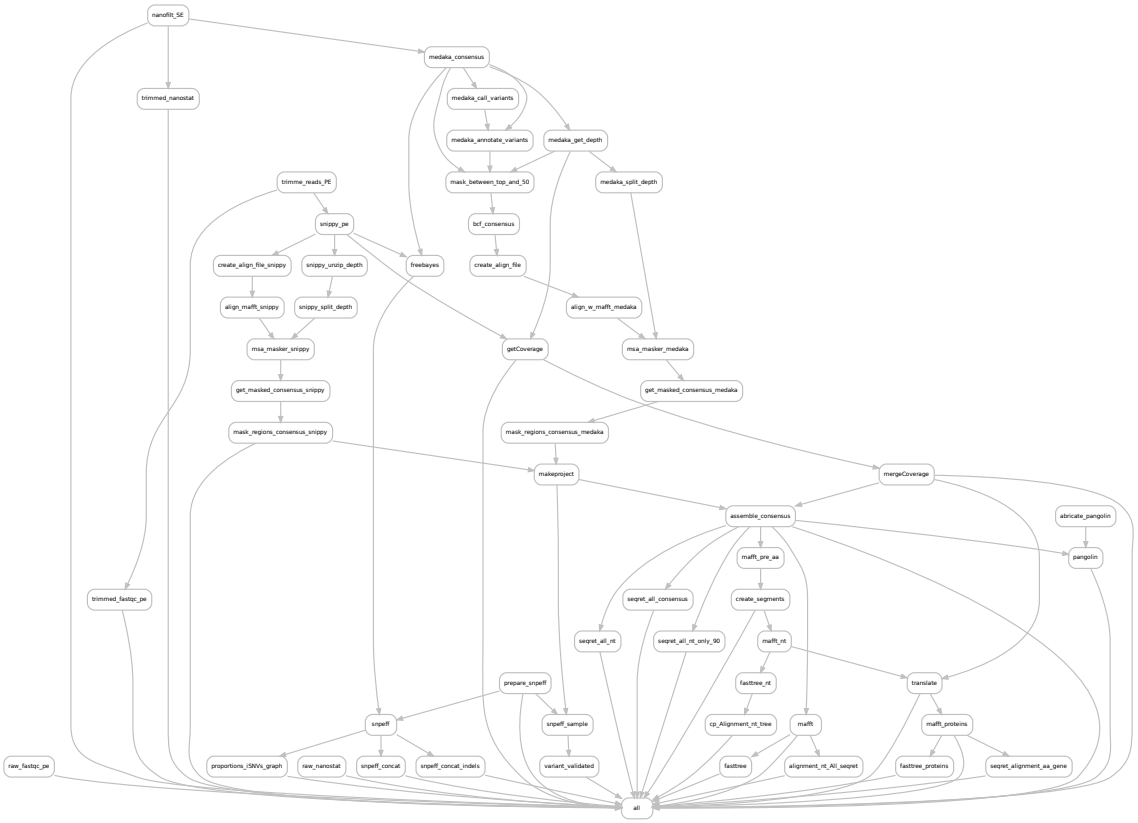

Figure 4.1: DAG of INSaFLU-Snakemake processing 2 samples. The DAG produced by Snakemake outlines the multi-step workflow for the analysis of 1 Illumina and 1 ONT samples.

tool to cross-verify matches against the SARS-CoV-2 reference. In the event of a match, the pipeline triggers the Pangolin software to conduct a lineage assignment of the provided samples. The second checkpoint considers the horizontal coverage of each sample. If any sample's coverage exceeds a user-defined threshold, the pipeline proceeds to the consensus-gathering phase, followed by sequence alignment and translation into the respective species proteins.

4.1.3 DAG and Rulegraph

Figures 4.1 and 4.2 show the different paths for Illumina and ONT samples as well as how the DAG expands horizontally with an increase in the number of samples. Figure 4.3 shows the rulegraph of the pipeline, which displays the dependencies between rules, independently of the number of samples.

4.1.4 Checking for Bottlenecks

I utilized Snakemake's benchmark utility to gather data concerning the computational resources expended by the pipeline's individual instances of rules. To provide a more

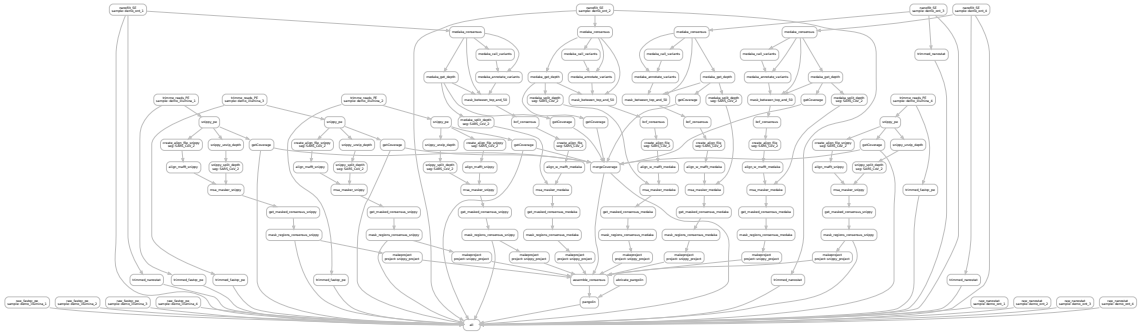

Figure 4.2: DAG of INSaFLU-Snakemake processing 8 samples. The DAG produced by Snakemake outlines the multi-step workflow for the analysis of 4 Illumina and 4 ONT samples.

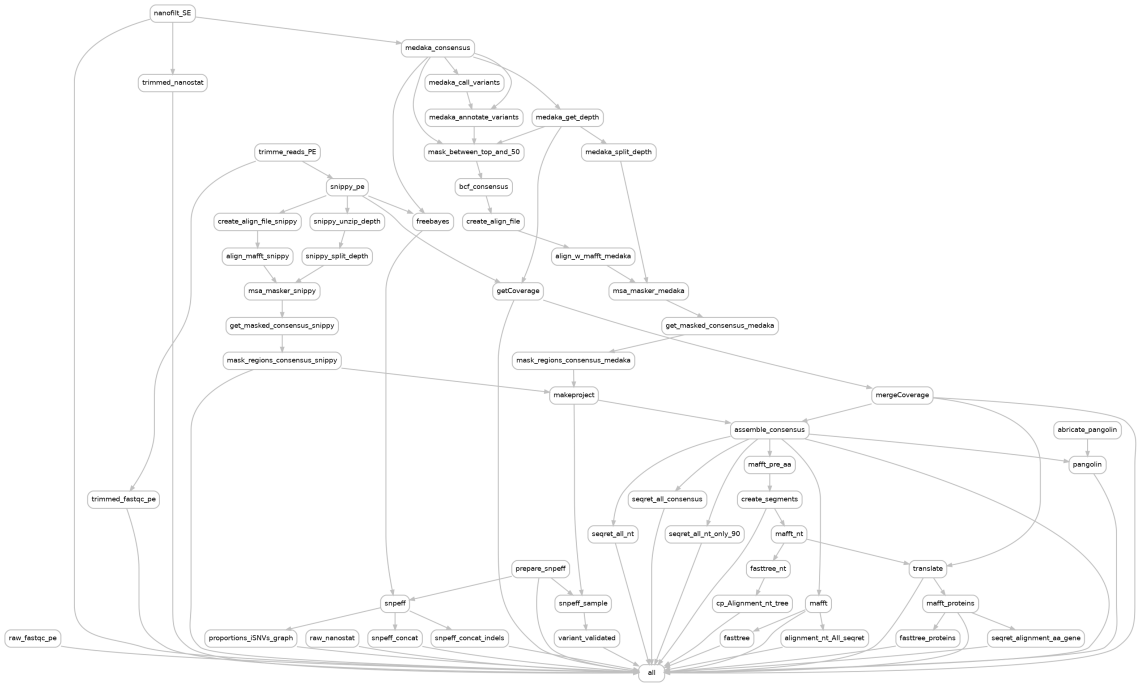

Figure 4.3: Rulegraph of INSaFLU-Snakemake processing Illumina and ONT samples on project mode.

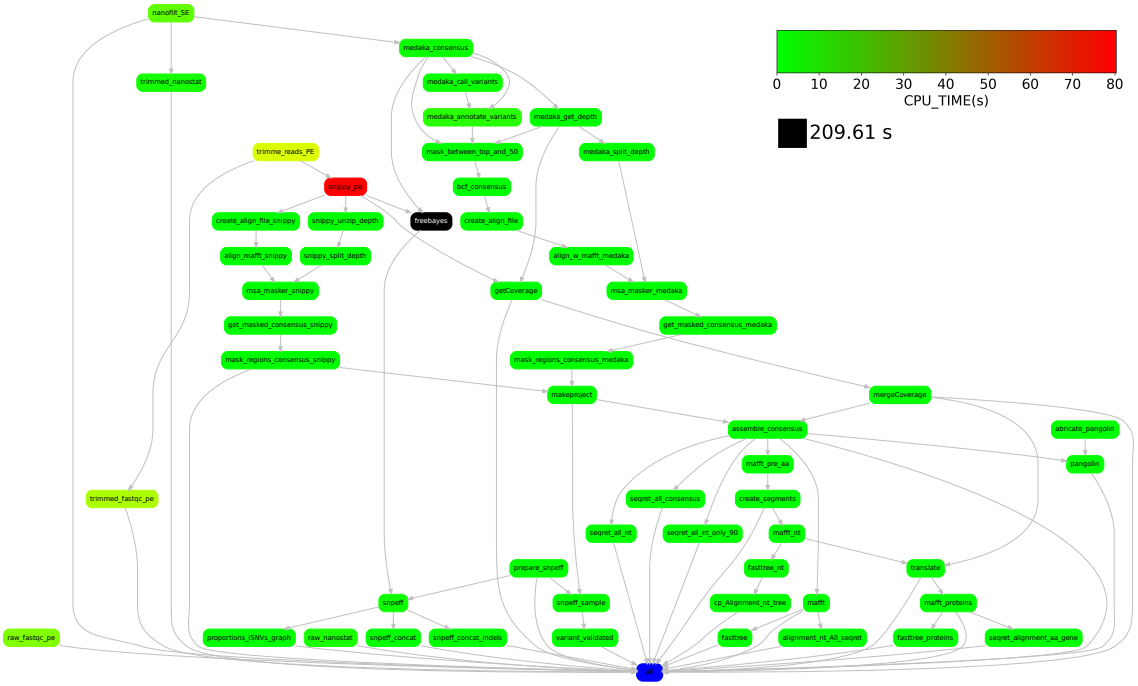

Figure 4.4: Rulegraph colored with mean CPU time by rule, analyzing 1 Illumina and 1 ONT sample from SARS-CoV-2. Note that the black node, which falls outside the gradient legend, was plotted separately due to being significantly out of scale. Additionally, the blue node represents the end of the pipeline and does not have an associated time.

comprehensible representation of this data, I developed a script that translated the resource metrics into corresponding color codes. These color codes were then applied to both the rulegraph (refer to Fig. 4.4 and Fig. 4.5) and the horizontal bar plot (refer to Fig. 4.6 and Fig. 4.7). This dual visual approach serves to effectively differentiate between the rules and their respective mean time taken for execution, as well as their relative resource usage. This strategy significantly enhances the potential for comparative visual analysis.

In the rulegraph and bar plots, it is possible to see that the rule that exhibits the highest CPU time consumption is the minor variant detection software, "freebayes," displaying a notable discrepancy when compared to the second one "snippy." Subsequently, there are "trimmomatic" and "fastqc," each followed by other rules. Notably, the remaining rules demonstrate a CPU time consumption of less than 15 seconds when analyzing two samples. Shifting the focus to memory usage, "snippy" stands out as the one prominent rule instance that demands the most memory resources. This is followed by "fastqc", "pangolin," "trimmomatic," and others at similar values.

4.1.5 Comparison between INSaFLU and Snakemake

I ran several samples from different organisms and technologies and compared the results obtained with INSaFLU and INSaFLU-Snakemake. All scripts used for the comparison

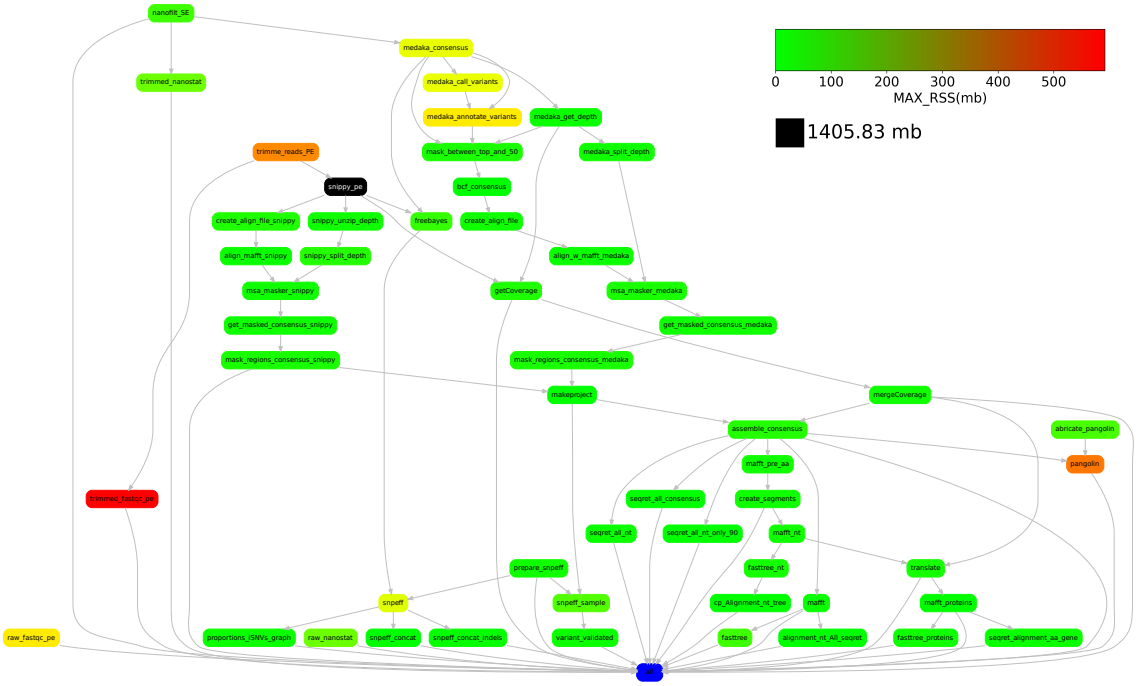

Figure 4.5: Rulegraph colored with mean used memory by rule analyzing 1 Illumina and 1 ONT samples from SARS-CoV-2. Note that the black node, which falls outside the gradient legend, was plotted separately due to being significantly out of scale. Additionally, the blue node represents the end of the pipeline and does not have an associated time.

can be found at `github.com/RdrigoE/compare_insaflu_snakemake` and rerun by following the instructions in the README.md file. The exact values of the comparison can be found in the table 4.4.

Out of the initial 40 samples collected for SARS-CoV-2, only 21 of them successfully met the coverage threshold. Within the 21 Monkeypox samples, only 16 passed the coverage threshold. Similarly, among the 13 Influenza samples, only 9 of them met the coverage threshold. It's important to note that Influenza differs from SARS-CoV-2 and Monkeypox in that its genome consists of 8 segments instead of 1. The 9 Influenza samples that met the coverage threshold, all 8 segments were successfully covered. Additionally, in 4 other samples, although not all segments met the threshold individually, a total of 20 segments from these samples did pass the threshold. In total, there were 92 segments present in the consensus, combining those from the 9 samples with all segments meeting the threshold and the 4 samples with some segments meeting the threshold.

It's worth noting that the reference is not included in the consensus. However, the alignment does include the reference, which is why the alignment has one more entry than the consensus.

The AllConsensus and Alignments results were 100% identical to those on the INSaFLU website. A discrepancy was noted only in the Alignment of Proteins when more than one segment was present in the gene, when the protein is constituted by only one

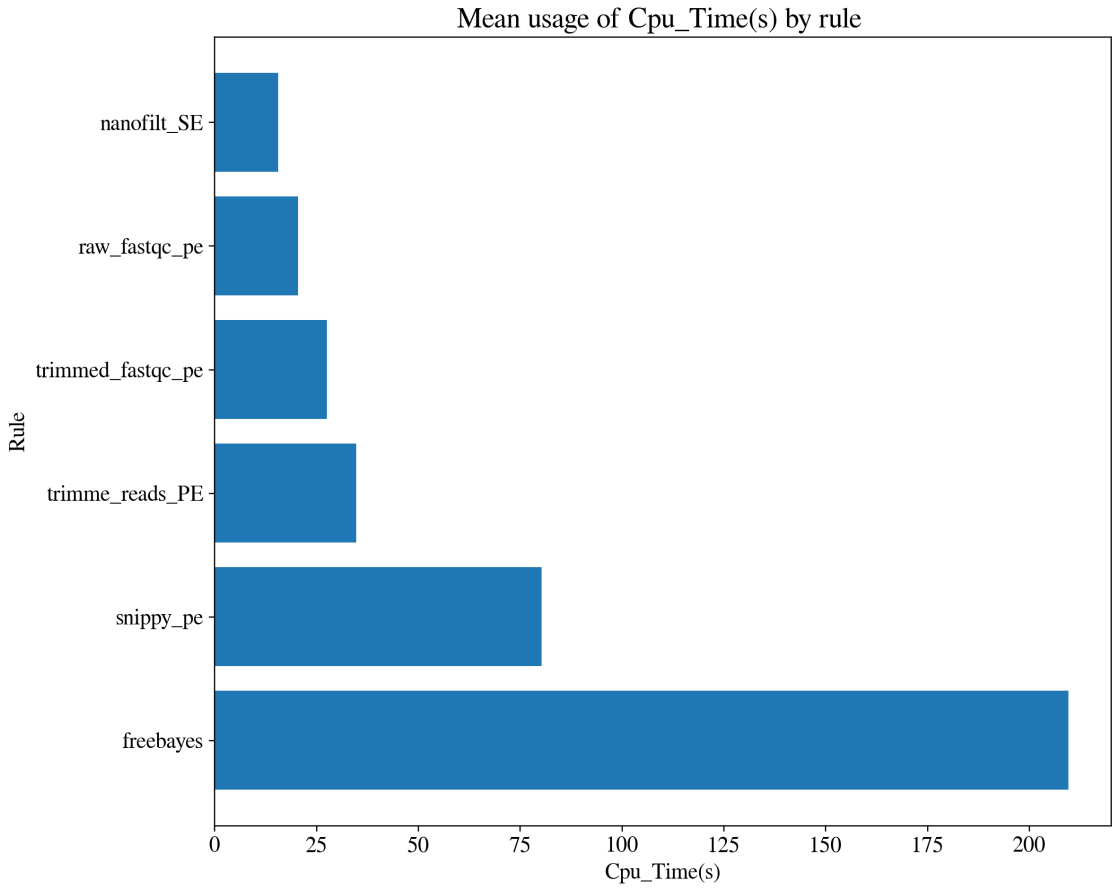

Figure 4.6: Horizontal Bar Plot of mean CPU time by rule for INSaFLU-Snakemake rules that take longer than 15 seconds in a run with 1 Illumina and 1 ONT sample.

Table 4.4: Description of matches and differences between INSaFLU and INSaFLU-Snakemake outputs.

Comparison	**SARS-CoV-2**		**Monkeypox**		**Influenza**	
	Match	**Differences**	**Match**	**Differences**	**Match**	**Differences**
Consensus	21	0	16	0	92	0
Alignment	22	0	17	0	-	-
Coverage	40	0	21	0	13	0
Clonal Variants	275	0	864	0	3576	48
Minor Variants	207	11	122	2	140	13

segment it has a perfect match.

The validated variants, that is, the variants with more than 50% frequency, were found to match 100% with the website for SARS-CoV-2 and Monkeypox and had 98.67% match of the total variants with Influenza, 9 out of 13 samples with a few different variants (Tab. 4.4). However, the minor variants showed a 5.04% difference from the website for Illumina SARS-CoV-2, 1.6% for Monkeypox, and 9.15% for Influenza. A 100% match was observed with the website results in the Coverage Analysis.

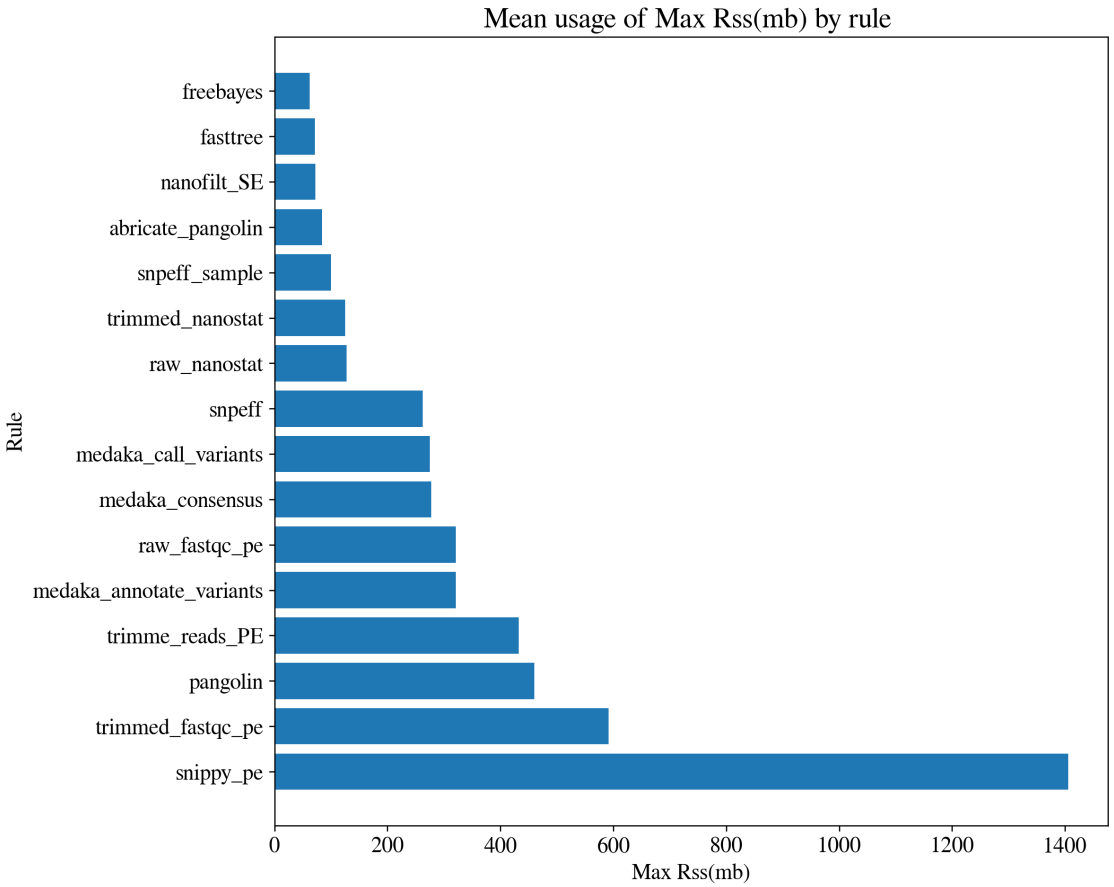

Figure 4.7: Horizontal Bar Plot of mean used memory by rule for INSaFLU-Snakemake rules that use more than 50 megabytes in a run with 1 Illumina and 1 ONT sample. Note that the black node, which falls outside the gradient legend, was plotted separately due to being significantly out of scale. Additionally, the blue node represents the end of the pipeline and does not have an associated time.

4.2 Extending INSaFLU-Snakemake

4.2.1 iVar module as an alternative to snippy

The iVar computational package is a versatile tool designed for viral amplicon-based sequencing. It combines functions from various tools to facilitate variant calling and generating consensus sequences. The key features of iVar include primer removal, enhancing data quality, and generating accurate consensus sequences by identifying dominant nucleotides at each position in the viral genome. It also excels in variant calling, detecting iSNVs, and insertions/deletions. iVar's capability to identify primer mismatches improves data accuracy by excluding misaligned reads. In essence, iVar provides an integrated solution for robust iSNV and consensus sequence calling in viral research and bioinformatics.

The iVar module is composed of 16 rules, 7 more than the snippy implementation.

To allow the choice between snippy and iVar some of the expected output was rewritten with a dynamic variable that is dependent on the user configuration for each run on the ”user/project.yaml” that can have the value ”snippy” or ”iVar”. To allow users to specify primers for iVar another key was added to the project configuration file, the ”primers_fasta”.

The expected outputs are outlined within the Snakefile. The initial module selection relies on retrieving a variable from the ”user/project.yaml” file. Subsequently, this variable is applied to generate the expected output as follows:

```
expand(
    "align_samples/{sample}/{Illumina_software}/{sample}_consensus.fasta",
    Illumina_software=Illumina_assembler,
    sample=single_Illumina.keys(),
)
```

The variable ”illumina_assembler” is passed to the subsequent module’s rules. This ensures that they possess the necessary information to retrieve files either from iVar or snippy.

4.2.2 Comparing iVar and Snippy

4.2.2.1 Real Samples

I compared the results of processing 1110 samples using two bioinformatics tools, iVar and Snippy, for variant calling. Figure 4.8 revealed that the median number of matches between the two tools was 29838, close to 20903, the genome length of SARS-CoV-2. The nucleotide mismatches between the two tools showed a mean and median of 0, indicating a generally high level of agreement (Tab. 4.5). However, in particular samples, up to 12 mismatches were observed (Tab. 4.7). Regarding Snippy Ns where iVar placed a nucleotide showed a mean and median of 0, indicating a minimal occurrence of this type of discrepancy, with only 70 samples with one or more occurrences. The maximum number of Ns placed by Snippy in a single sample was 40. On the other hand, iVar placed Ns where Snippy had a nucleotide, with a median of 3 and a mean of 75, in 415 samples it had 0 occurrences and in 593 had 1 or more Ns. Regarding the placement of gaps and Ns, where iVar had an N and Snippy had a gap, the mean was 17 and median was 13, and the maximum value was 284. Conversely, when Snippy had an N and iVar had a gap, the mean was 3 and the median was 0, with a maximum value of 282. Regarding the placement of gaps and nucleotides, Snippy placed a gap where iVar had a nucleotide, showing a mean of 2 and a maximum of 29 occurrences in a single sample. When iVar placed a gap where Snippy had a nucleotide, the mean was 0, except for two outliers with 2 and 11 occurrences.

Table 4.5: Statistics on mismatched per sample and coverage.

Condition	Number of samples
Samples	1110
No coverage	102
0 mismatch	833
1 mismatch	142
>1 mismatch	33

Table 4.6: Statistics on 1110 samples processed by INSaFLU-Snakemake with Snippy and iVar.

statistic	median	mean	variance	min	max
matches	29838	29764.18	183.33	28824	29853
mismatch	0	0.3	1.07	0	12
snippy_conservative	0	0.74	3.81	0	41
ivar_conservative	3	74.56	170.76	0	954
gap_snippy	2	2.08	1.83	1	29
gap_ivar	0	0.01	0.35	0	11
snippy_gap_ivar_N	13	17.2	28.99	5	284
snippy_N_ivar_gap	0	3.58	28.96	0	282
uncovered_case	0	0.02	0.14	0	2

Table 4.7: Distribution of mismatch counts among 1110 SARS-CoV-2 samples.

Number of mismatches	Number of samples
0	833
1	142
2	10
3	2
4	3
5	5
6	5
7	4
8	1
12	3

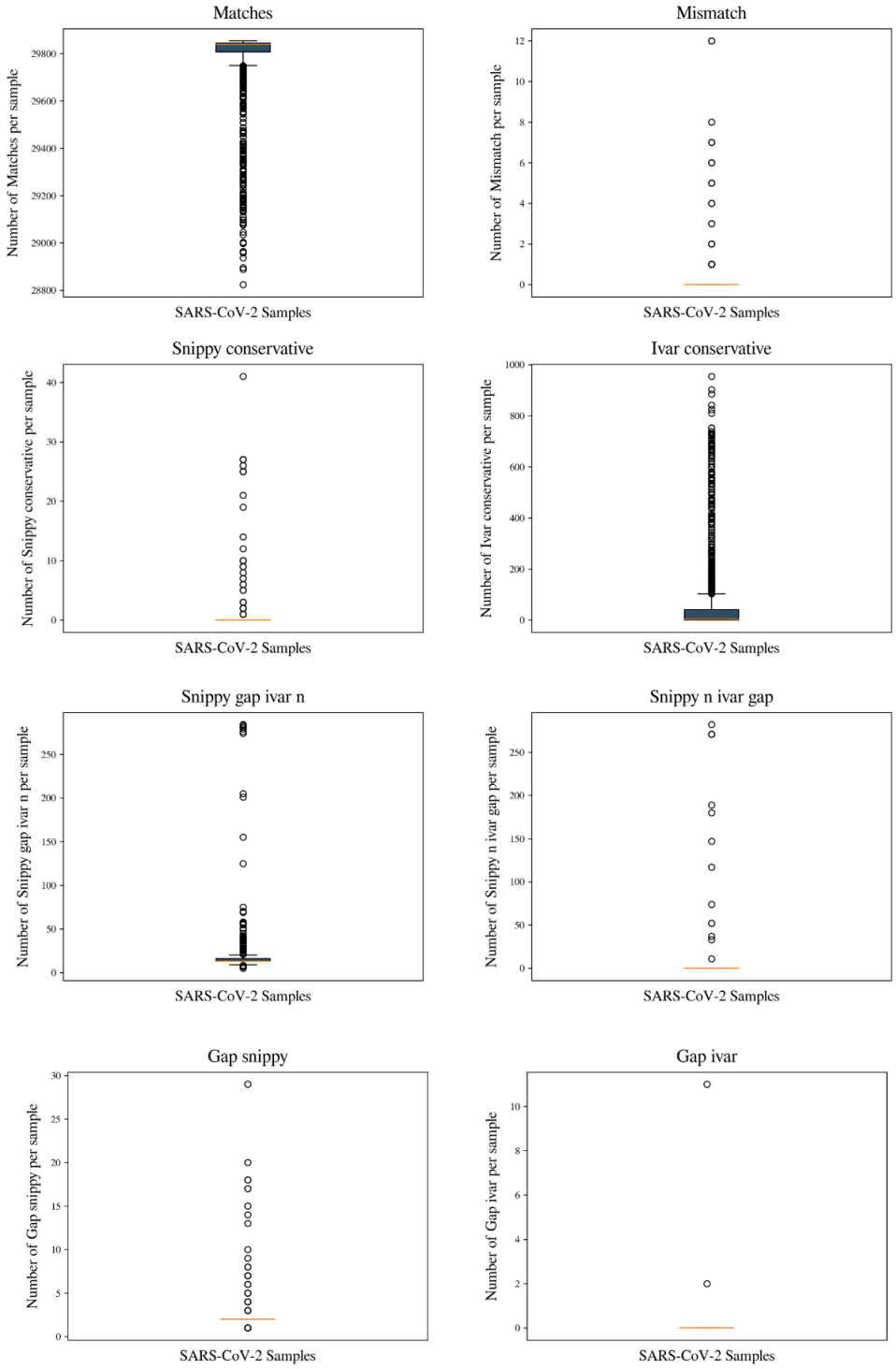

Figure 4.8: Comparison between snippy and iVar using 1110 SARS-CoV-2 samples. Eight plots show different aspects of variant detection: Matches, Mismatches, Conservative snippy and iVar, Snippy with gaps and iVar Ns, Snippy with Ns and gaps iVar, Gap snippy and Gap iVar.

4.2.2.2 Increasing SNPs in artificial samples

In this study, 12 different SNP conditions were tested, ranging from 0 to 1000 SNPs, each condition involving 100 samples. These samples were subjected to analysis using two tools, snippy and iVar. The results, as depicted in Figure 4.9, reveal a notable trend: as the number of SNPs increases in the artificial samples, the median accuracy of both iVar and Snippy consistently remains at 100%. This finding underscores the robust performance of these tools across a wide range of SNP conditions. However, 1 sample from 400, 1 from 500, and 7 from 1000 SNPs condition exhibited slightly lower accuracy, ranging from 68% to 74% accuracy. The median number of Ns had a noticeable difference from the median number of Ns for the samples processed by iVar and Snippy. The number of Ns varied by around 20 across all conditions, suggesting potential differences in their ability to handle missing data, uncertain bases, or genomic positions. Both iVar and Snippy achieved a median number of mismatches equal to 0 for all conditions. This indicates that both tools were highly effective in accurately detecting sequence variations in the artificial samples.

4.2.2.3 Decreasing Spike Identity in artificial samples

In the study, eight different identity frequency conditions were examined, varying from 100% down to 50%, with each condition comprising 100 samples. These samples were analyzed by both snippy and iVar. With the figure 4.10, it is possible to see that by decreasing the identity level at Spike Gene at higher identity percentages (above 85% - 573 SNPs), both iVar and Snippy maintained a 100% accuracy rate. However, as the identity decreased, the accuracy of both tools started to decline. When the identity reached 50% (1910 SNPs), the accuracy dropped to 0%, indicating a complete loss of precision in variant calling. For iVar, the median number of mismatches remained consistently at 0 for all identity levels, highlighting its robustness in maintaining accuracy irrespective of sequence similarity. For Snippy, the median number of mismatches increased as the identity decreased. At 80% identity, it rose to 1 mismatch, increasing to 2 mismatches at 75% identity before eventually dropping to 0 at 50%. The number of Ns in the samples notably increased as the identity percentage decreased from 85% to 50%. At 50% identity, the median number of Ns reached 3500. The experiment revealed that both iVar and Snippy exhibited high accuracy in variant calling at higher sequence identity and even when handling many SNPs.

4.2.3 Edge Cases

Snippy and iVar have different algorithms and the main difference should be noticeable in cases near primers. The following tables showcase the SNP frequency at different genomic positions in SARS-CoV-2 and Monkeypox genomes and the performance of Snippy, Snippy with Trimmomatic, and iVar in detecting variants near primer regions. In the case of the SARS-CoV-2 genome at position 22789 (Tab. 4.8), where the reference nucleotide

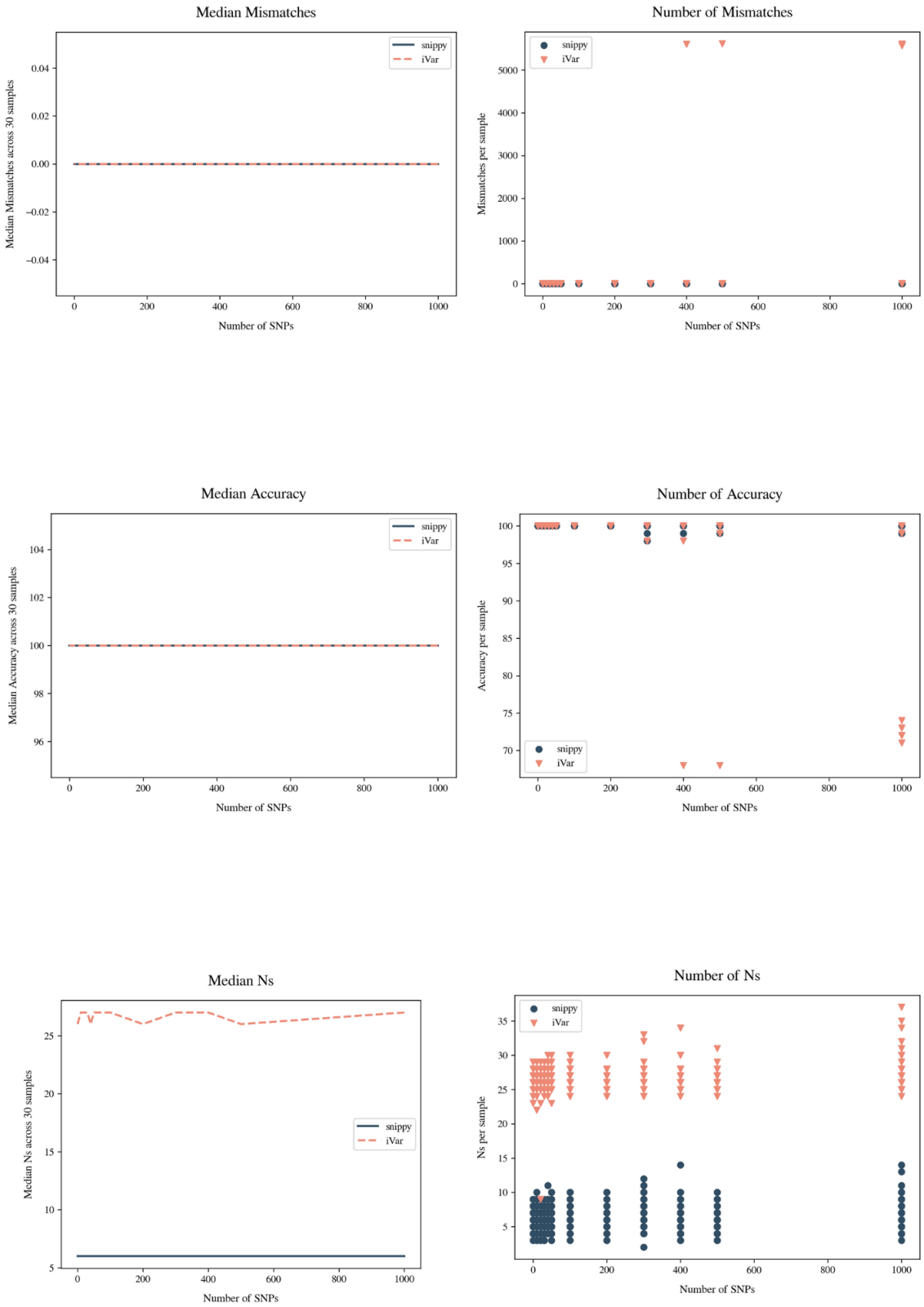

Figure 4.9: Snippy and iVar performance using artificial samples with an increasing number of SNPs. On the right is shown the median mismatch, accuracy, and Ns, on the left are the respective scatter plots.

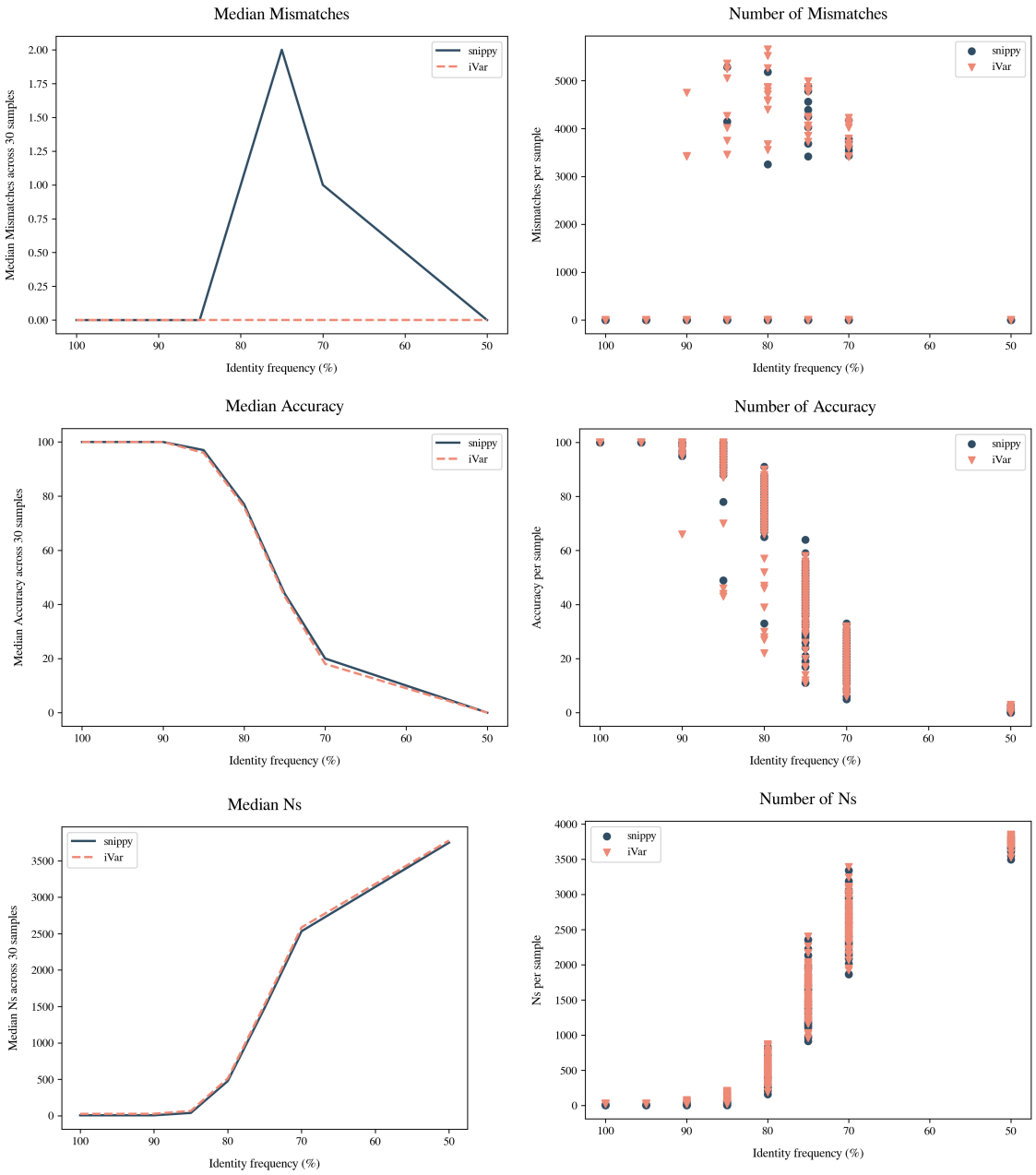

Figure 4.10: Snippy and iVar performance using artificial samples with decreasing identity at Spike gene. On the right is shown the median mismatch, accuracy, and Ns, on the left are the respective scatter plots.

Table 4.8: SNP frequency in SARS-CoV-2 Genome at position 22789, reference nucleotide A, altered nucleotide C. Analysis with snippy, with snippy with primer trimming in Trimmomatic and iVar. The variant should be clonal.

Sample (22789 A>C)	Snippy(wo/ primers)	Snippy(w/ primers)	iVar
Portugal-PT45141-2022	0.43	0.376	0.9259
Portugal-PT45265-2022	0.31	0.207	0.9736
Portugal-PT45116-2022	0.37	0.243	0.9391
Portugal-PT44134-2022	0.32	0.245	0.9379
Portugal-PT44766-2022	0.32	0.257	0.977

Table 4.9: SNP frequency in Monkeypox Genome at position 13068. Analysis with snippy, with snippy with primer trimming in Trimmomatic and iVar. The SNP should be removed, iVar, and snippy with trimmomatic corrected snippy.

Samples	Snippy (wo/primers)	Snippy(w/primers)	iVar
164	1	-	-
191	0.67	-	-
198	0.79	-	-

A is changed to C, the results demonstrate that iVar consistently outperforms Snippy and Snippy with Trimmomatic, calling the SNP as clonal for all tested samples. This indicates iVar's ability to successfully identify and call primer-proximal variants, making it a suitable tool for accurately detecting such mutations. Similarly, in the Monkeypox genome at position 13068 (Tab. 4.9), the correct form is not to call the SNP, and both iVar and Snippy with Trimmomatic perform well, fixing the variant with SNP frequencies reduced to 0 for all samples. In contrast, Snippy without primer trimming shows a high SNP frequency for sample 164, indicating that the primer-proximal variant remains unresolved, highlighting the importance of primer trimming before variant calling. Furthermore, at position 13090 (Tab. 4.10) in the Monkeypox genome, iVar and Snippy with Trimmomatic effectively remove the SNP, resulting in SNP frequencies 0 for all samples. However, the Monkeypox genome at position 12836 (Tab. 4.11) presents a challenging scenario. Here, the goal is to remove the SNP from the first sample and consider it clonal in the second sample. None of the tools achieve the desired outcome, as SNP frequencies remain present for both samples. This illustrates the complexity of primer-proximal variant calling, where all three tools consistently struggle to resolve such cases.

4.3 Enabling execution in a SLURM cluster

Executing INSaFLU-Snakemake on the SLURM is easily done via a single node, where it runs like a local execution. To take advantage of the parallelization that Snakemake and the pipeline offer, running the resource intensive tasks in multiple nodes greatly increases the pipeline's execution speed.

Table 4.10: SNP frequency in Monkeypox Genome at position 13090. Analysis with snippy, with snippy with primer trimming in Trimmomatic and iVar. The SNP should be removed, iVar, and snippy with trimmomatic corrected snippy.

Samples	Snippy (wo/primers)	Snippy(w/primers)	iVar
164	0.76	-	-
182	0.58	-	-
183	0.6	-	-
186	0.63	-	-
191	0.52	-	-
198	0.93	-	-

Table 4.11: SNP frequency in Monkeypox Genome at position 12836. Analysis with snippy, with snippy with primer trimming in Trimmomatic and iVar. The SNP should be removed from the first sample and be clonal in the second one. None of them fixed it.

Samples	Snippy (wo/primers)	Snippy(w/primers)	iVar
347	0.81	0.86	0.85
290	0.2	0.249	0.2

4.3.1 Analyzing INSaFLU workflow using Snakemake tools

Snakemake provides command line flags to generate rulegraphs, as demonstrated earlier. In the upcoming section, I will highlight discrepancies in resource consumption while running identical samples and parameters. This analysis will encompass scenarios involving both a single node setup and a SLURM cluster. To enhance clarity, I will utilize horizontal bar plots along with color-coded rulegraphs to visually depict these variations.

4.3.1.1 Rulegraph and Detection of bottlenecks

The mean use of the resources by each rule in the 50 Monkeypox samples are described in the rulegraphs (Fig. 4.11 and 4.12) and the Horizontal plots 4.13,4.14, 4.15.

As it is possible to understand from the mentioned rulegraph and bar plots, the rule/-software that has the higher mean of CPU time in SLURM and single node is mafft, fastree and trimmomatic. The remaining rules use less than 15 CPU seconds. The CPU time is equal in SLURM and single node. Looking at the memory, the one rule/software that has more mean use of memory is fastree, then mafft, snippy, followed by trimmmomatic. Another parameter exhibiting a noticeable discrepancy is the time taken from rule initiation to completion. The execution on a single node needs more time to finalize the rules compared to the SLURM implementation.

4.3.2 Running INSaFLU in the INCD SLURM-based cluster

The INCD (Infraestrutura Nacional de Computação Distribuída) in Portugal offers researchers shared access to powerful computing resources. This boosts scientific progress

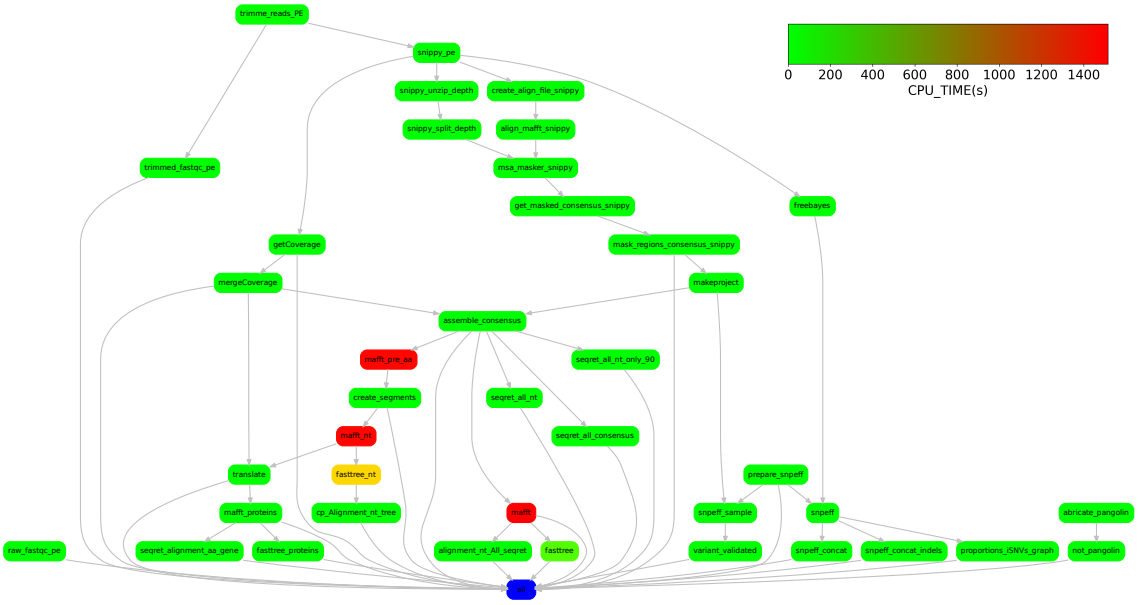

Figure 4.11: Rulegraph colored with mean CPU time by rule analyzing 50 Illumina from Monkeypox in a single node. The blue node represents the end of the pipeline and does not have an associated time.

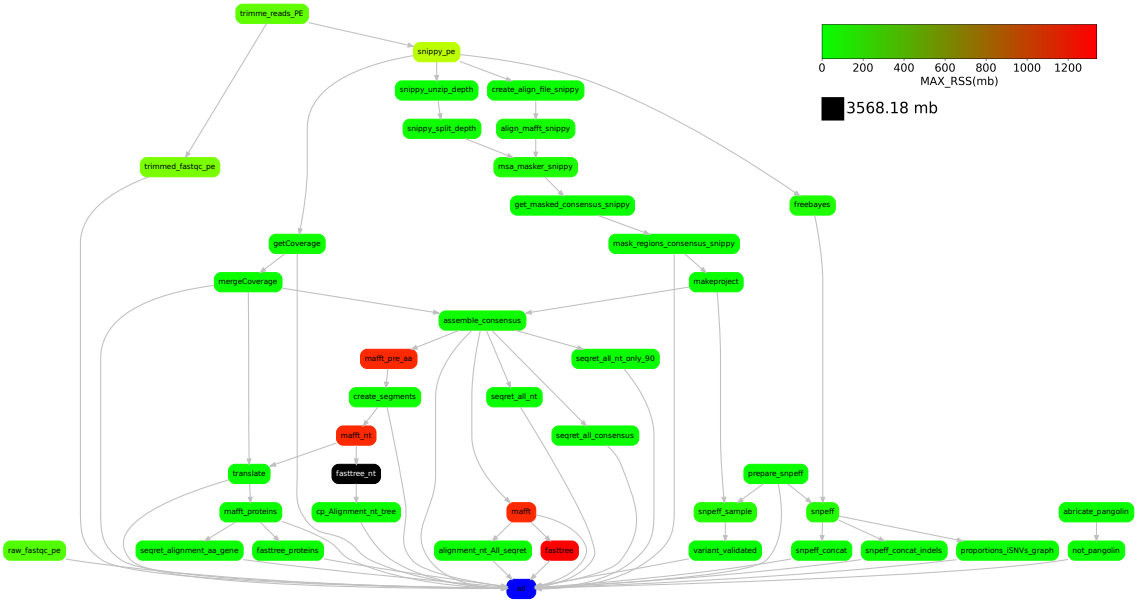

Figure 4.12: Rulegraph colored with mean used memory by rule analyzing 50 Illumina from Monkeypox in a single node. Note that the black node, which falls outside the gradient legend, was plotted separately due to being significantly out of scale. Additionally, the blue node represents the end of the pipeline and does not have an associated time.

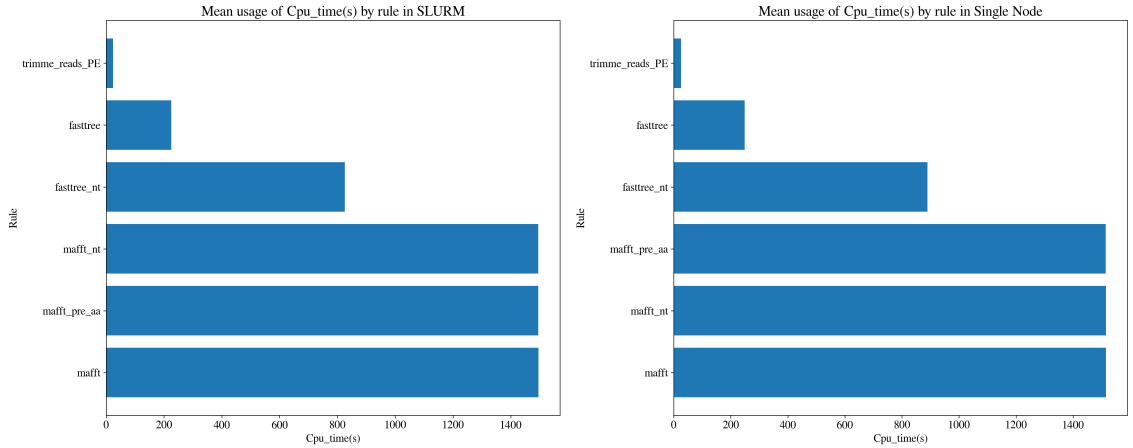

Figure 4.13: Horizontal Bar Plots of mean CPU time by rule for INSaFLU-Snakemake rules that take longer than 15 seconds in a run with 50 Illumina samples in SLURM (on the right) and a single node (on the left).

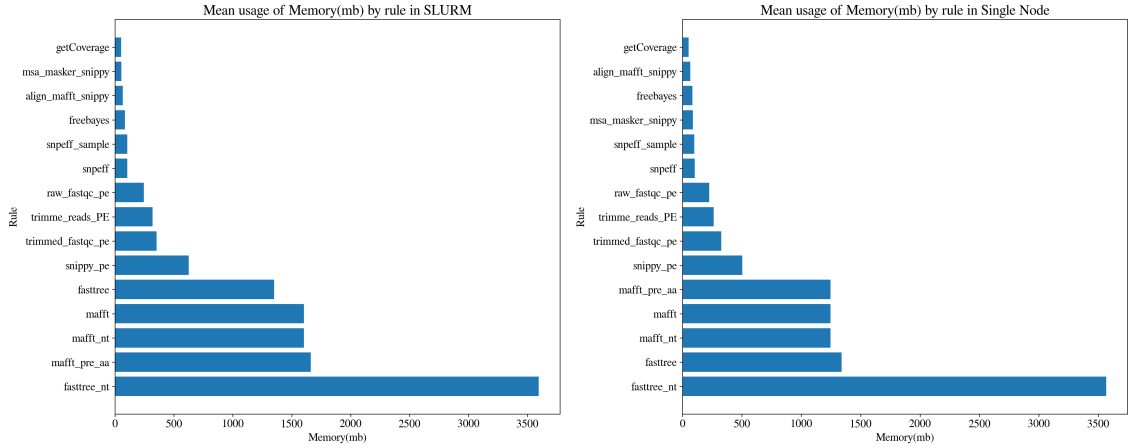

Figure 4.14: Horizontal Bar Plots of mean used memory by rule for INSaFLU-Snakemake rules that use more than 50 megabytes in a run with 50 Illumina samples in SLURM (on the right) and a single node (on the left).

by enabling complex simulations and data analysis across various fields. This thesis was supported by Instituto Nacional de Computação Distribuida (INCD) through project 2022.23037.CPCA.A0 where I had dedicated resources. The partition where the work was done has 37 nodes in the partition. Each node has 16 CPUs, and each CPU has 8 cores, which together are great for running tasks in parallel. These nodes also have plenty of memory, with a capacity of 30,000 MB, which is helpful for tasks that need a lot of memory.

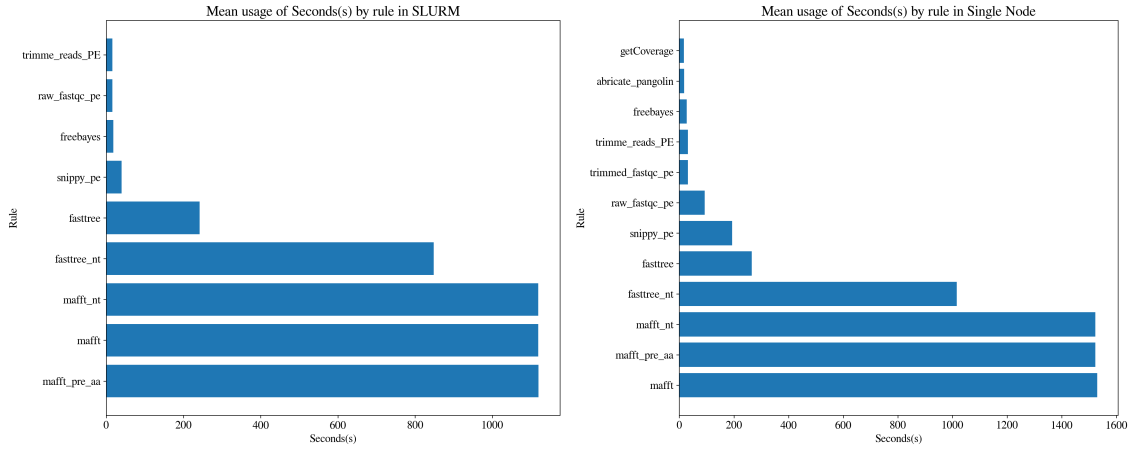

Figure 4.15: Horizontal Bar Plots of mean execution time by rule for INSaFLU-Snakemake rules that take longer than 15 seconds in a run with 50 Illumina samples in SLURM (on the right) and a single node (on the left).

4.3.3 Execution in SLURM cluster

In the SLURM environment, there are three steps to be run. Editing of slurm/config.yaml and slurm/run_slurm.sh with the required user's parameters.

This is the example file of "slurm/config.yaml"

```
cluster:
  mkdir -p logs/{rule} &&
  sbatch
    --partition={resources.partition}
    --qos={resources.qos}
    --cpus-per-task={threads}
    --mem={resources.mem_mb}
    --job-name=smk-{rule}-{wildcards}
    --output=logs/{rule}/{rule}-{wildcards}-%j.out
default-resources:
  - partition=<name>
  - qos=<name>
  - mem_mb=1000
restart-times: 3
max-jobs-per-second: 10
max-status-checks-per-second: 1
local-cores: 1
latency-wait: 60
jobs: 500
keep-going: True
rerun-incomplete: True
printshellcmds: True
scheduler: greedy
use-conda: True
```

This is the example file of "slurm/run_slurm.sh"

```
#SBATCH --output /path/to/insaflu_snakemake/profile-%j.out
#SBATCH --error /path/to/insaflu_snakemake/profile-%j.err
#SBATCH -p {partition}
#SBATCH -q {queue}

# activate conda in general
source path/to/.bashrc # to conda init setting

# activate a specific conda environment, if you so choose
conda activate /path/to/mambaforge/envs/insaflu

# go to a particular directory
cd /path/to/insaflu_snakemake

# make things fail on errors
set -o nounset
set -o errexit
set -x

### run your commands here!

snakemake --profile slurm
```

The files should be filled with the partition and queue, and the run_slurm.sh moved into the main folder, "insaflu_snakemake". After that run the following command:

```
sbatch slurm/run_slurm.sh
```

The queued jobs can be seen with the following command:

```
squeue -u <user_name>
```

4.3.4 Compare single-node and cluster

Running in a single node and cluster should only have a resource and time of execution difference, and that was the case. The first notable difference was in the execution time. The SLURM took only 1 hour and 33 minutes, while the single node took 6 hours. The CPU time is the same for both platforms (Fig. 4.13), but the time it takes to finish a rule is always less in the SLURM (Fig. 4.15). This is due to the way that Snakemake handles the jobs. When they are sent to the cluster, they have their own node to run on with defined resources. When using a single node, the resources need to be shared by all active jobs. The memory used in the SLURM per rule is also higher (Fig. 4.14). It is possible to see notable differences from the analysis with two samples to this one, the rules that used more CPU time were project related, in this case, it was mafft, the software used to align all the consensus followed by fastree, used to create tree files from the aligned consensus. The sample processing rules appear after these. The rule that used more memory was fastree followed by mafft and snippy. It is important to understand that although mafft

and fastree are the one's using more resources they only run a couple of times while, trimmomatic, fastqc, snippy, medaka run on each sample. Projects with a lot of samples benefit from parallelization especially in the sample processing part as it's possible to see in figure 4.2.

4.4 Showcase INSaFLU-Snakemake by analyzing the Portuguese Monkeypox dataset

4.4.1 Characterize SNPs

Analyzing the Monkeypox dataset using the INSaFLU-Snakemake produced insights into the genetic variation among samples. The number total of variants was 35778. The clonal variants were 21107. From these, 19547 were annotated. It is possible to see in figure 4.16 that most of these were missense variants, making up 57.64%. Missense variants are interesting as they involve changes in a single nucleotide that result in the coding of a different amino acid, potentially impacting protein function. Synonymous variants, accounting for 42.24% of the total, are also notable as they involve changes that do not alter the protein's amino acid sequence and can provide valuable information about the evolution and functional constraint of the viral genomes. For SNPs with a frequency between 10% and 50% has 2162 variants in total among 492 samples but only 1,074 are annotated, the majority were also missense variants (58.06%), followed by frameshift variants (16.59%). The number of SNPs with a frequency less than 50% was 14671 with 11288 annotated. A large portion of these SNPs were missense variants (56.93%) and frameshift variants (22.65%).

4.4.2 Distribution of SNPs in the genome

The results section of this study identified regions of the Monkeypox genome with the highest density of variants for different frequency ranges. These regions of heightened SNP density may indicate genomic hotspots for mutation and may hold significant implications for viral evolution and function. The genomic region showing the highest SNP density for clonal SNPs was identified between the genome's start and position 7640, which are part of the inverted terminal repeat of Monkeypox (Fig. 4.17). Other regions demonstrating increased SNP concentration were between positions 34,380-38,200 and 68,760-72,580. In the case of SNPs with frequencies ranging between 10% and 50%, the region spanning from positions 145,160-148,980 exhibited the highest SNP density. Additional areas with high SNP concentration included the regions from positions 126,060-129,880 and 30,560-34,380. Considering the 11,278 SNPs with a frequency below 50%, the genomic region from positions 126,060-129,880 had the highest SNP density.

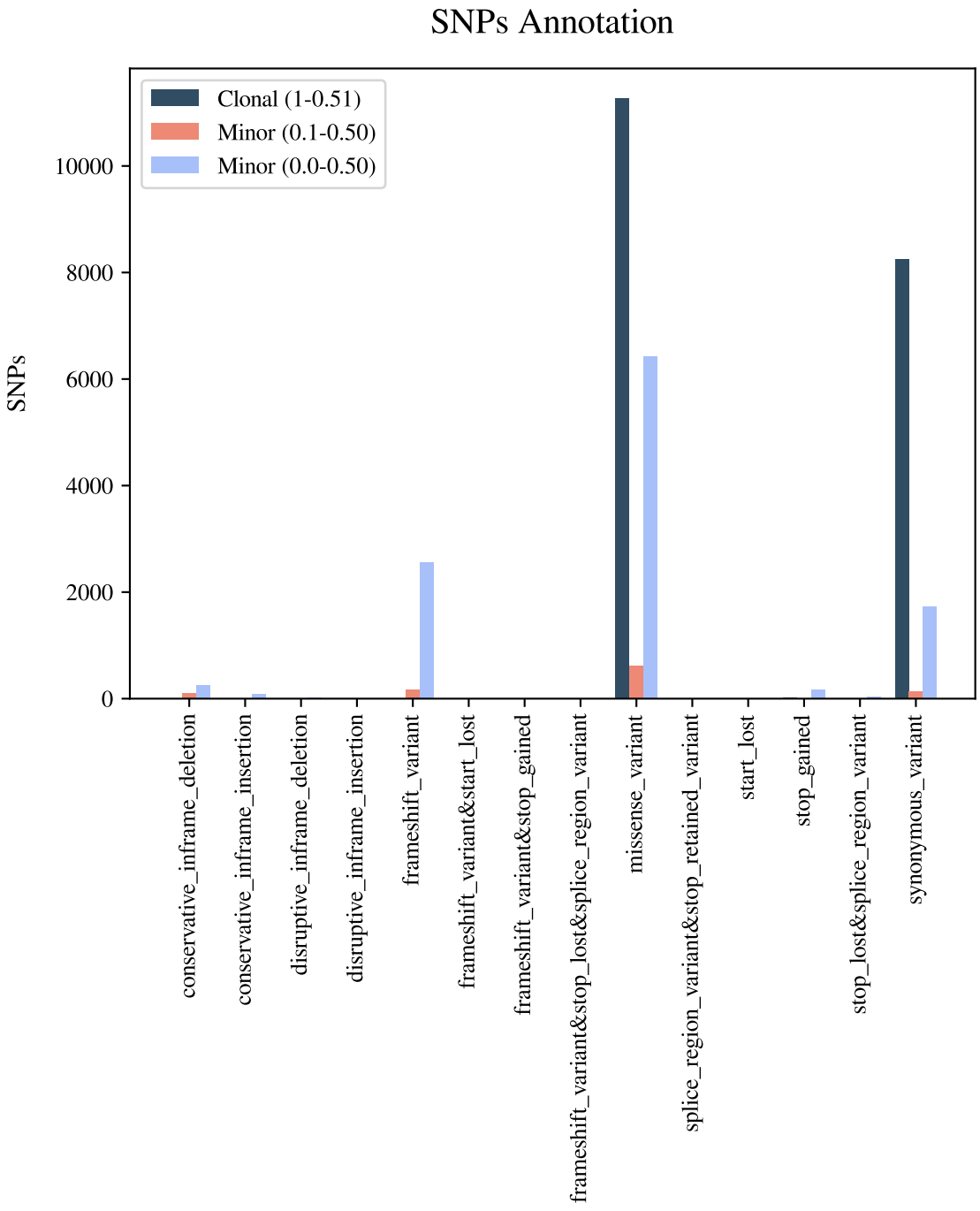

Figure 4.16: Annotation of SNPs from 500 Monkeypox Samples analyzed by INSaFLU Snakemake with snippy at different variant frequency ranges (1-0.51; 0.1-0.50;0.0-0.50)

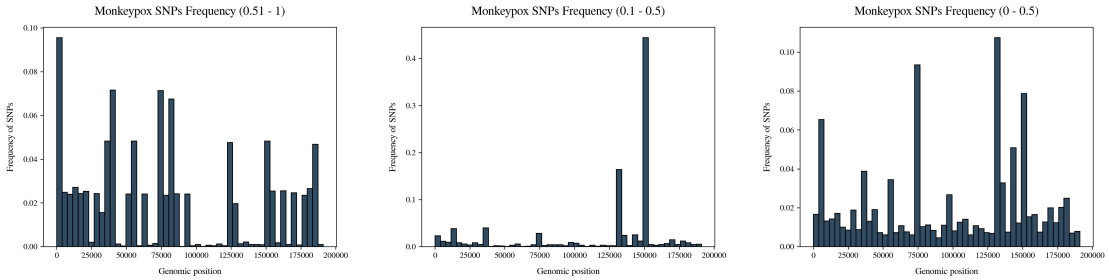

Figure 4.17: Distribution of SNPs from 500 Monkeypox Samples analyzed using INSaFLU-Snakemake using snippy at different variant frequency ranges (1-0.51; 0.1-0.50;0.0-0.50).

4.4.3 Comparing results with what is known in the literature

All the Isidro et al. SNPs were present in the study (Fig. 4.19). From the 41 SNPs, 38 of them were present in at least 90% of the samples and the remaining 3 were between 60% and 85% of the samples.

The APOBEC3 enzyme family plays a crucial role in the body's innate immune response to viral infections by introducing mutations into the viral DNA. This is achieved by deaminating cytosine to uracil in single-stranded viral DNA, resulting in AG > AA e TC > TT mutations [44]. This enzymatic activity can lead to the accumulation of mutations within the viral genome, potentially impairing viral replication and thus controlling the spread of the infection.

APOBEC mutations (Fig. 4.18) were detected across different ranges: 0.51 - 1, 0.1 - 0.5, and 0.0 - 0.5. The purpose behind examining the fraction 0.51-1 was to highlight clonal variants, while the range of 0.0-0.50 aimed to encompass all minor variants. Additionally, the interval of 0.1-0.50 was specifically chosen to effectively filter out noise arising from minor variants. It is worth noting that the minor variants have a minimum coverage of 100. In the 0.51 - 1 range, a total of 18,440 APOBEC mutations (87.3%) were identified alongside 2,667 non-APOBEC mutations. Within the 0.1 - 0.5 range, 346 APOBEC mutations were observed, along with 1,816 non-APOBEC mutations. In the lower 0.0 - 0.5 range, 1,519 APOBEC mutations were present alongside 13,152 non-APOBEC mutations.

4.4.4 SNPs present in 90% of the samples

When analyzing the dataset with snippy, 38 clonal SNPs were present in at least 90% of the samples (Fig 4.20). Remarkably, 33 of these (approximately 87%) were identified as APOBEC mutations. This high proportion of APOBEC-related mutations suggests a strong involvement of APOBEC enzymes in shaping the viral genetic diversity as part of the host's immune response to the Monkeypox virus. The remaining five SNPs, representing about 13% of the clonal SNPs found in 90% of the samples, were not associated with APOBEC activity. These mutations might be attributable to other mutagenic processes. The annotation of those SNPs were two synonymous variants and one missense

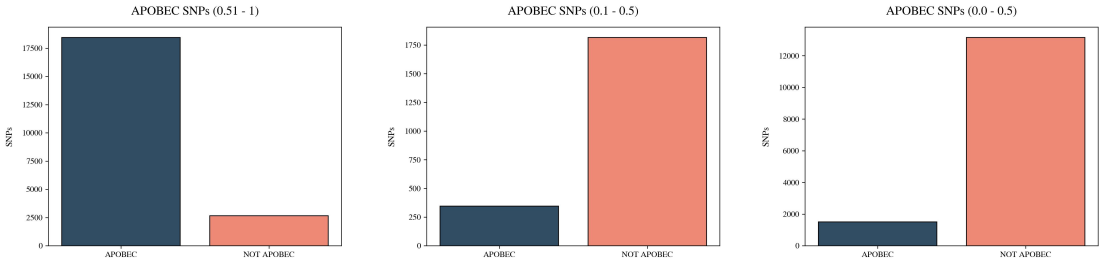

Figure 4.18: Number of APOBEC and NON-APOBEC SNPs from 500 Monkeypox Samples analyzed by INSaFLU Snakemake with snippy at different variant frequency ranges. 1-0.51 with 18,440 APOBEC and 2,667 non-APOBEC mutations; 0.1-0.50 with 346 APOBEC and 1,816 non-APOBEC mutations; 0.0-0.50 with 1,519 APOBEC and 13,152 non-APOBEC mutations.

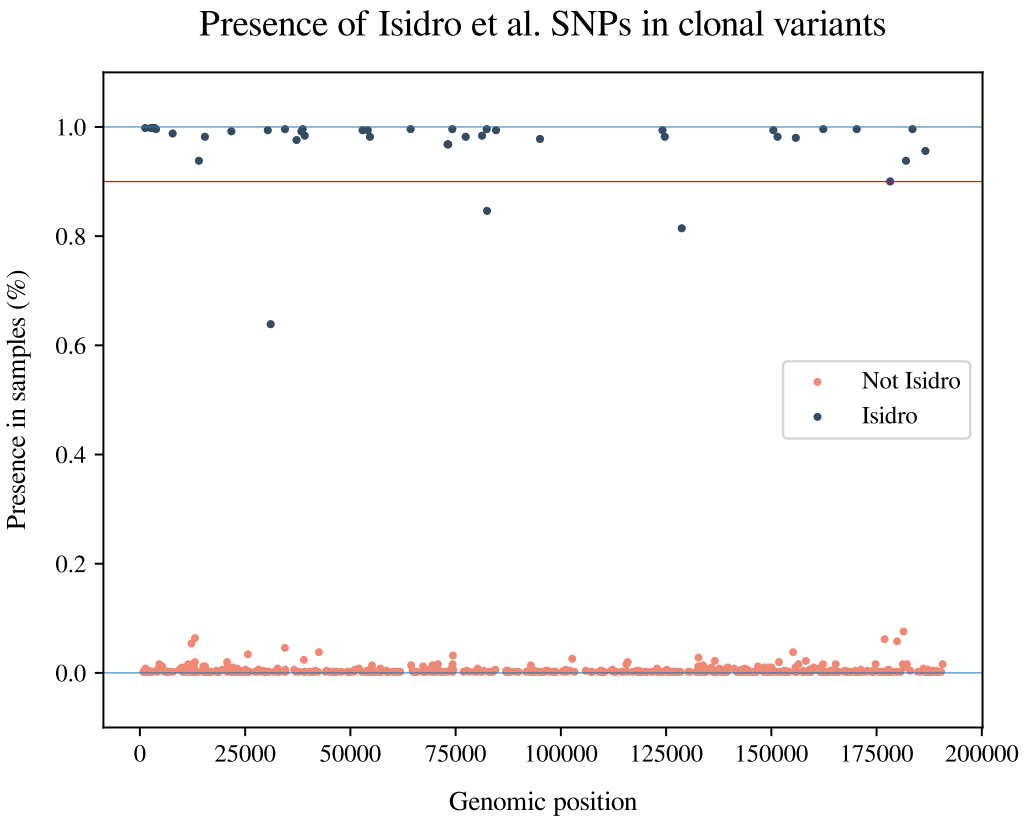

Figure 4.19: Plot with SNPs percentage of presence in samples and genomic position using snippy. Color-coded according to their annotation, blue dots represent the SNPs present in Isidro et al. study. Red line represents the 90% sample presence.

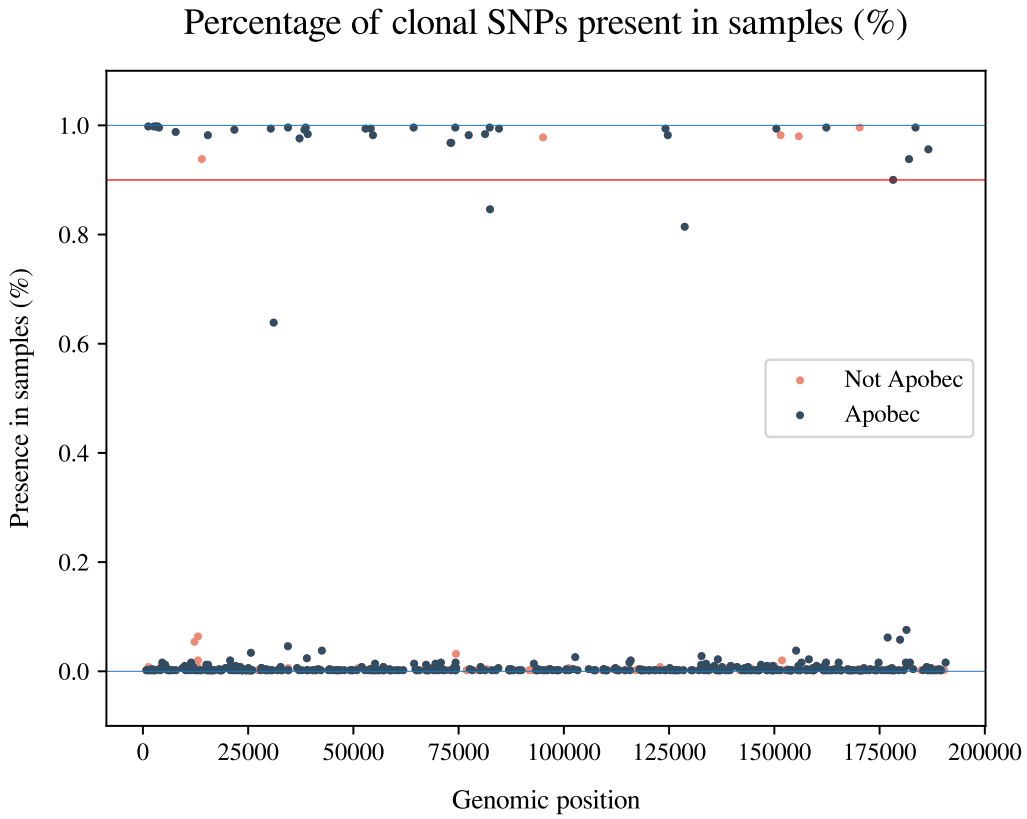

Figure 4.20: Plot with SNPs percentage of presence in samples and genomic position using Snippy. Color-coded according to their annotation, APOBEC in blue and non-APOBEC in pink.

variant. The remaining had no annotation. These findings provide valuable insights into the interaction between the Monkeypox virus and its host's immune system, particularly the role of APOBEC enzymes. When analyzing with iVar, some variants decrease their presence in the dataset, as it is possible to see in figure .5, there are only 36 SNPs within 90% or more samples. One of the APOBEC and one non-APOBEC variant decreased their presence below the 90% limit. All the annotated non-APOBEC variants above 90% sample presence processed by iVar were synonymous variants.

4.4.5 Correlation between hamming distance and time

The goal of integrating minor SNPs into each Monkeypox sample's consensus sequence was to reveal more intricate relationships between the samples. This approach hypothesized that such integration would amplify differences between samples and therefore enhance the correlation between the genetic distance of the samples and their collection times, which would reflect in the structure of the phylogenetic tree. However, the results from this experiment did not meet the expected outcome. The correlation values between the genetic distance and collection time failed to improve significantly with the

Table 4.12: Correlation between genetic distance and time of collection

Condition	Correlation (genetic distance and time)
Snippy	0.29
Minor 10% Snippy	0.28
Minor Snippy	0.23
iVar	0.28
Minor 10% iVar	0.28
Minor iVar	0.26

introduction of minor SNPs. As the table 4.12 displays, the original correlation value for genetic distance and collection time using the Snippy clonal SNPs was 0.29. After integrating minor SNPs (at different frequency thresholds), the correlation values dropped to 0.28 and 0.23 for above 10% frequency and all minor SNPs, respectively, suggesting that adding minor SNPs did not strengthen the correlation but slightly decreased it. Similar results were observed with the iVar data. The original correlation value was 0.28, which remained at 0.28 for SNPs above 10% frequency and decreased to 0.26 for all minor SNPs. Again, this indicates that introducing minor SNPs did not enhance the correlation. These results imply that integrating minor SNPs into the consensus sequences did not improve the resolution of the phylogenetic relationships among the Monkeypox virus samples in relation to their collection times. Thus, the experimental strategy failed to achieve its intended goal of uncovering more complex relationships among the samples through minor SNPs. This finding suggests that other factors not accounted for in the current analysis might play a more significant role in the evolution and divergence of the Monkeypox virus over time.

4.4.6 SNPs differences between snippy and iVar

To verify differences between snippy and iVar a chi-square test was performed to verify if the number of SNPs identified by the two software significantly differ. Both of them share 20867 SNPs, iVar has 35, and snippy has 269 unique SNPs. The null hypothesis is the number of SNPs identified by Snippy and iVar is not significantly different and the alternative is that there is a significant difference between the number of SNPs identified by the two software tools. Upon conducting the chi-square test, the calculated test statistics were 0.0000, and the p-value was 1.0000. The extremely low test statistics and the p-value of 1.0000 suggest no significant differences. In other words, no evidence indicates that one software is more effective in finding SNPs. Therefore, this study has no statistical basis for favoring one software for finding SNPs.

5

Discussion

5.1 INSaFLU-Snakemake

The replication of INSaFLU was a success as all the functionality was implemented with some minor differences that can be improved in the future. This implementation can be run easily on local systems, SLURM and it's also ready for other platforms.

5.1.1 Snakemaking INSaFLU challenges

Recreating INSaFLU using Snakemake posed significant challenges due to the project's complexity and the diversity of action paths dependent on parameters and the input data, the output is determined mid-execution by specific data characteristics amplifying the intricacy of the task. Despite these challenges, the expressiveness of Snakemake, such as checkpoints, the ability to execute scripts post-conda environment download, and global variables, proved invaluable during the development of the initial version of INSaFLU Snakemake.

INSaFLU's documentation was a vital asset in the early stages of development, providing helpful information about software versions and parameters, significantly aiding the creation of straightforward rules. However, once this led to a false path only identified when comparing the two pipeline implementations. Namely, the coverage produced by INSaFLU and INSaFLU-Snakemake were clearly different, to the point that some samples were passing the consensus threshold and others were not. This led to significant differences in the output. After analyzing the problem I verified that the documentation was pointing to a GitHub repository with a Python script and the INSaFLU repository was using an altered version of the code on that repository, and that was not documented. As is often the case, the source code served as the ultimate point of reference where all definitions reside. Despite its utility, navigation within the source code was occasionally challenging, especially since the functionality is frequently abstracted behind classes, and Python's typeless nature can complicate the assessment of property and parameter types within each class and their associated methods.

5.1.2 Comparison with CWL

The CWL employs a fundamentally declarative execution methodology. It uses an execution engine to interpret the workflow description and organize task scheduling. Conversely, Snakemake, heavily integrated with the Python programming language, adopts a more procedural strategy based on explicit rules. CWL and Snakemake support Conda and Docker, facilitating effective dependency management. However, Snakemake's integration with the previous tools and Python is deeper, often making it a preferable choice for Python programmers. The selection between the two largely hinges on personal preference and specific user requirements. While some users might value the language-agnostic nature of CWL, others might lean towards the Python-centric approach of Snakemake. Pipeline complexity, portability needs, and user familiarity with the tools can influence the choice between CWL and Snakemake. However, both CWL and Snakemake have powerful features and serve as robust alternatives for workflow management, each offering distinct advantages depending on the specific context. Snakemake workflows can be exported to CWL, enabling execution by any executor. However, due to Snakemake's more expressive Python integration, some patterns may be deconstructed into smaller steps to replicate the same output in CWL. For this project, Snakemake's export feature is impractical as it involves checkpoints, which CWL currently does not support. The lack of real-time DAG rearrangement in CWL, unlike Snakemake, makes conversion infeasible. Using checkpoints does not compromise the predictability and repeatability of Snakemake's implementation. Providing the same inputs will consistently generate the same outputs. The checkpoints are solely data-dependent, meaning the same data will always yield the same output.

5.1.3 Why different outputs Snakemake - INSaFLU

Ideally, one would anticipate a 100% match between all files across the two implementations. However, this was not the case, attributable to differences in Python scripts, especially in protein translation and possibly in software versions in variant calling. For Validated and Minor Validated Variants, the discrepancies likely originate from the variant calling software, freebayes, that currently does not have the version INSaFLU uses in conda or GitHub. Regarding the translation process, the current INSaFLU implementation is encapsulated within several classes. As previously mentioned navigating through the code base can be challenging, especially when the implementation is abstracted behind layers of classes. The disparities highlighted in table 4.4 regarding validated influenza variants do not influence the consensus since they are in complete agreement. This phenomenon emerges from the method of variant description. The variants were defined based on the following factors: position, chromosome, reference nucleotides, and altered nucleotides. The dissimilarities are determined by the differences in any value of the factors described, for example, the count of nucleotide variations, such as A to T is different from AA to AT, or position, from 20001 to 20000. Despite the consensus being

identical, these distinctions are considered variations due to the calculation methodology. The specific presentation of these dissimilarities possibly varies due to the minor difference in the 'snippy' version used by INSaFLU-Snakemake and INSaFLU.

5.2 Extending INSaFLU-Snakemake

The study strongly emphasized the concept of modularity, a fundamental idea in this thesis. Snakemake facilitates modularity due to its close relation with Python. Python allowed the preprocessing of the samples and parameters to configure the execution in the Snakefile, with this it was easier to set up dynamic output files and these trigger the right rules in the pipeline. Snakemake has a notably modular approach, the investigation revealed a dynamic and adaptable structure that efficiently enabled the swapping and replacement of key parts. This modular setup, exemplified by the option between Snippy and iVar for detecting genetic variations, highlighted the framework's flexibility and ability to explore different approaches thoroughly. Beyond adding flexibility, this modular method sets a valuable example for efficient experimentation.

The study's approach to comparing software alternatives further highlights its rigorous methodology. By switching between Snippy and iVar, it was possible to assess how well various genetic variant detection tools performed within a unified framework. The broad range of measurements used to assess variant calling accuracy and quality across real and artificial samples showcases the study's strong commitment to thorough analysis. These methods make the study more credible and emphasize the detailed attention given to understanding the strengths and limitations of the new module. Combining modularity and well-thought-out evaluation techniques creates a unified and impactful research effort that drives progress.

5.2.1 Snippy and iVar

Accurately and efficiently detecting genetic variations is a critical aspect of research. This study compared two bioinformatics tools, iVar and Snippy, for variant calling in real and artificial samples. The results shed light on the performance and discrepancies between the two tools, providing valuable insights for variant calling strategies in different genomic contexts. Overall, the study demonstrated that both iVar and Snippy showed a high level of agreement in variant calling for most samples, as indicated by the number of matches close to the total genome length of SARS-CoV-2. This suggests that both tools are reliable for variant detection in routine scenarios. However, it is essential to acknowledge that some samples exhibited variations and mismatches, which could have implications for specific genomic regions or challenging samples. One notable difference between the two tools was their handling of ambiguous bases (Ns) and consensus gaps. Snippy showed minimal occurrences of Ns, indicating a less conservative algorithm. In contrast, iVar exhibited some occurrences of Ns placed where Snippy had nucleotides, suggesting

that it is more conservative than snippy. These observations highlight the importance of understanding how each tool handles ambiguous data and the potential impact on variant calling accuracy. The performance of both iVar and Snippy in artificial samples provided valuable insights into their ability to handle increased numbers of SNPs and decreased sequence identity, although the artificial samples are not appropriate for iVar as they simulate a genome instead of amplicons. Both tools maintained high accuracy when dealing with increased SNPs, showcasing their robustness in detecting variants in such scenarios. However, an anomaly was detected among 9 specific samples out of 1200. These samples exhibit accuracy levels ranging between 68% and 74% within the context of 400, 500, and 1000 SNPs. Notably, these specific samples consistently display an unusual quantity of mismatches, between the range of 5000 to 6000. This divergence in mismatch frequency becomes evident when contrasted with the broader dataset. It is noteworthy to highlight that these particular samples exhibit a consistent pattern of Ns. However, the presence of mismatches of such a high magnitude suggests the alignment process with the reference genome as the primary contributing factor. The edge cases analysis revealed differences between iVar and Snippy in detecting primer-proximal variants. iVar consistently outperformed Snippy and Snippy with Trimmomatic in resolving primer-proximal variants in certain genomic positions, highlighting its potential advantages in such regions. This emphasizes the importance of selecting appropriate tools based on specific genomic contexts, especially in primer regions, to ensure accurate variant calling results. iVar has an advantage in pipeline incorporation due to the main functionality being distributed between multiple steps giving the user the option to generate the files independently. Snippy is more strict, it has a lot of parameters but it is a one-package software where all action happens under the hood and it may lead to a more difficult integration in a pipeline. Several factors may contribute to the observed differences between iVar and Snippy. Algorithmic dissimilarities, variations in handling ambiguous bases and gaps, and user-defined parameters are some of the potential reasons for the discrepancies. It is crucial to be aware of these factors and consider them when selecting a variant calling tool for specific research tasks. The biological results and implications between the two software will most of the time favor iVar over snippy but it can happen that iVar silences a variation.

5.2.2 Integrating iVar and Snippy to find problematic cases

iVar's main distinction comes from its use of primer information to filter out some reads that could introduce erroneous data. However, this strategy could mask correct information, so case-by-case verification is necessary. When both snippy and iVar are run on the same sample, it is possible to find nucleotide mismatches where the changes from the reference are present in snippy consensus. Running multiple samples and checking the mismatched positions often reveals common issues. To validate these, the files bam pre and post-primer trimming should be used. A usual pattern is the substantial presence of the snippy nucleotide at one end of the reads, which are eliminated afterward. This could

indicate an over-amplification of a modified primer. An extensive set of samples should be processed using both software to pinpoint these situations more efficiently, with the consensus sequences aligned and the mismatches marked. Next, all the mismatches are grouped by position and nucleotides. If a position appears repeatedly, it warrants further investigation, which can be facilitated with the help of the bam files.

5.3 Enabling execution in a SLURM cluster

5.3.1 The Speed Advantage of SLURM

SLURM, which stands for Simple Linux Utility for Resource Management, allows for extensive parallelization, especially when paired with a dedicated queue. The essence of this speed advantage lies in SLURM's architecture and its compatibility with Snakemake. A key feature of SLURM is its ability to efficiently allocate resources and manage workloads, making it highly conducive for running tasks in parallel. This parallel execution capability is particularly enhanced when SLURM is combined with Snakemake. Snakemake, designed with a focus on flexibility and scalability, can distribute multiple independent tasks across the available nodes in a SLURM cluster, meaning that many tasks can be processed simultaneously rather than sequentially, drastically reducing the total execution time. Moreover, Snakemake's ability to handle dependencies further optimizes the execution process. It allows the system to skip completed tasks and only run those necessary, which saves considerable time, especially when pipelines are rerun or modified. In a dedicated queue scenario, jobs typically experience minimal wait time before execution begins, further speeding up the process. With a dedicated queue, resource contention is significantly reduced, and jobs can start immediately after submission, resulting in faster overall execution times. This is seen when running the 50 Monkeypox samples, the total duration decreased from 6 hours to 1 hour and 33 minutes due to the high parallelization.

5.3.2 Resource Management

Snakemake provides the capability for specifying resource requirements, such as memory and threads, within a workflow. These parameters are flexible, allowing users to tailor resource allocation according to their specific dataset and computational needs. Notably, the resource demands can vary significantly when working with different organisms, as exemplified by the contrasting requirements of SARS-CoV-2 and Monkeypox. Monkeypox, characterized by a larger genome, necessitates greater memory allocation for certain rules due to the unique software demands associated with it.

To address these varying resource requirements effectively, I have created a script hosted on GitHub (https://github.com/RdrigoE/get-memory-insaflu-snakemake/). This script performs a comprehensive scan of a benchmark folder and compiles the findings into a YAML file. Within this file, there is a comprehensive list of rules, each accompanied

by its recommended memory values. These values have been set to slightly exceed the maximum memory usage observed by each rule.

This tool equips users with the ability to fine-tune memory allocations for future executions, leveraging insights gained from previous runs. It's important to note that the current memory values are optimized for a dataset comprising 500 Monkeypox samples. Consequently, they are designed to comfortably accommodate this sample size while offering a significant margin for handling much larger datasets of smaller genomes, such as SARS-CoV-2.

5.4 Showcase INSaFLU-Snakemake by analyzing the Portuguese Monkeypox dataset

5.4.1 Characterize SNPs

The 500 Monkeypox samples were successfully run using the INSaFLU-Snakemake on the INCD cluster, using both snippy and iVar. This accomplishment demonstrates the capability of running the pipeline in a custom execution environment. The characterization of SNPs in the Monkeypox dataset provides valuable insights into the virus's genetic variation and evolutionary dynamics during the outbreak in Portugal in 2022. Understanding the nature and distribution of SNPs is crucial for unraveling the underlying genetic changes that drive viral diversity, potential functional impacts, and interactions with the host immune system. The study classified SNPs into three categories based on their frequency in the samples: clonal SNPs (frequency >50%), minor SNPs with frequency between 10% and 50%, and minor SNPs with frequency <51%. The abundance of clonal SNPs, which accounted for a significant portion of the total SNPs, indicates the presence of genetically related strains circulating during the outbreak. These clonal SNPs were found to be predominantly missense variants, leading to changes in a single nucleotide that potentially impact the coding for a different amino acid. In this case, it is most likely that the new amino acid has the same characteristics as the previous one. Additionally, the study identified synonymous variants among the clonal SNPs, mutations that do not change the amino acid sequence. While they may not directly affect protein function, synonymous variants can be valuable markers for understanding viral evolution and functional constraints. The presence of synonymous variants in the dataset suggests that the virus is under selective pressure to maintain specific genetic sequences, possibly to preserve essential functions. One of the confirmations in the study was the high proportion of APOBEC-related mutations among the clonal SNPs. APOBEC3 enzymes are part of the body's innate immune response to viral infections, introducing mutations in viral DNA and potentially impeding viral replication. APOBEC-related mutations indicate that the host's immune system actively responds to the Monkeypox virus during the outbreak. This finding highlights the dynamic interplay between the virus and the host's immune defense mechanisms. It suggests that APOBEC-mediated editing plays a critical role in

shaping the genetic diversity of the virus during the outbreak. Overall, the characterization of SNPs in the Monkeypox dataset provides crucial insights into the genetic landscape of the virus during the outbreak. The abundance of clonal SNPs, the prevalence of missense variants, and the presence of APOBEC-related mutations all point towards ongoing viral evolution and adaptation. Understanding the genomic hotspots for mutation and the potential functional impacts of specific SNPs can further our knowledge of the Monkeypox virus's pathogenesis, host interaction, and transmission dynamics. This information is invaluable for public health efforts, including developing practical diagnostic tools, therapeutics, and preventive measures to control future outbreaks. Furthermore, the study's approach to characterizing SNPs can serve as a valuable model for investigating genetic variation in other infectious disease outbreaks and understanding the complex dynamics of viral evolution in response to the host immune system.

In the Monkeypox study, it was expected that specific SNPs would be found based on previous knowledge in the field. The anticipated SNPs included 46 variants identified in the Monkeypox outbreak genome sequences. Among these, 24 were predicted to be non-synonymous, while 18 were expected to be synonymous. It was anticipated that a considerable proportion of the SNPs found would align with the previously identified variants, thus confirming the involvement of APOBEC3 and the specific mutation patterns GA > AA and TC > TT. A substantial presence of APOBEC-related mutations was observed during the characterization of SNPs, particularly among the clonal SNPs. Approximately 87% of the signature SNPs in at least 90% of the samples were identified as APOBEC-related, in line with the expected results However, it was also noted that other SNPs not related to APOBEC activity were present in the dataset. These non-APOBEC SNPs represented 13% of the clonal SNPs in 90% of the samples. These non-APOBEC-related SNPs are considered a regular occurrence in viral genomes, as they can result from various mutagenic processes. The study's findings on APOBEC-related and non-APOBEC-related SNPs in the Monkeypox dataset confirmed the initial expectations. The coexistence of different mutagenic processes highlighted the complexity of viral evolution and the genetic diversity that emerges during an outbreak. The analysis revealed that within the first 50 SNPs that occur most frequently across samples, this dataset encompasses the 41 SNPs featured in the Isidro et al. dataset. This congruence underscores a consistency between the two datasets, confirming that the SNPs identified in this study are reliable and accurate. The observed occurrence of a particular SNP from the study by Isidro et al., present in only 64% of the samples within this investigation, can be attributed to the SNP's specific genomic position, 31062. This genomic position consistently demonstrates a low level of sequencing read coverage. Among the total of 500 genetic samples under analysis, a mere 321 samples possess sequencing coverage that either matches or exceeds the established threshold of 30 reads. This particular circumstance emerges as at least one contributing factor to the unexpected prevalence of Isidro et al. SNP within a diminished proportion of the analyzed sample set.

A recent study by O'toole et al. on the Monkeypox viral clade IIb found that 90.8%

of the variants were APOBEC. Of these, 38.7% were missense variants, altering the protein sequence, while 35.7% were synonymous, not affecting the protein sequence. This thesis Monkeypox results, also identified a high proportion of APOBEC-related mutations among the SNPs and a high presence of missense and synonymous variants which are consistent with the Rambaut study's findings. Overall, the Rambaut study's contribution to understanding APOBEC-related mutations in viral genomes complements and strengthens the knowledge gained from the Monkeypox study.

5.4.2 Minor Variants

The experiment involving the integration of minor SNPs into the consensus sequences to uncover more intricate relationships among the Monkeypox samples did not yield the expected results. The hypothesis was that by incorporating minor SNPs, the genetic distance between samples and their collection times would exhibit a more robust correlation, leading to a more refined phylogenetic analysis. However, the results showed that introducing minor SNPs did not significantly improve the correlation between genetic distance and collection time. The correlation values remained relatively unchanged or slightly decreased when minor SNPs were integrated into the analysis. This outcome suggests that other factors may play a more significant role in the evolution and divergence of the Monkeypox virus over time beyond what was accounted for in the current analysis. One possible contributing factor could be the interval between the time of infection and sample collection, which was not considered in this experiment. Viral evolution and transmission dynamics are complex processes influenced by various factors, such as host immune responses, selective pressures, and viral replication rates. The temporal dynamics of the outbreak and the duration of viral replication within hosts could substantially impact the observed genetic distances between samples. Thus, additional temporal data, such as the interval between infection and sample collection, could lead to a more accurate and refined correlation analysis. Despite the lack of significant improvement in correlation values, it is essential to acknowledge that negative results like these can still contribute to scientific knowledge. Understanding what does not work or what factors do not contribute significantly can help researchers refine their methodologies and guide future investigations. Additionally, incorporating more extensive datasets and considering more sophisticated analytical approaches may shed light on the viral population's complex relationships and evolutionary patterns.

Although the use of minor genetic variations didn't contribute significantly to better defining relationships between samples, a clear observation can be made. Specifically, within the genomic range of 0.1 to 0.5, a distinct cluster of minor SNPs is noticeable in the region from 145,160 to 148,980 nucleotides. This particular genetic segment is where the A41L gene [45] is, which encodes a 30 kDa glycoprotein. While this glycoprotein isn't necessary for virus replication, it does play a crucial role in influencing how the host responds to infection. One of its notable functions is being secreted to limit the infiltration

of inflammatory cells into the infected area.

In this section, the main objective was to showcase the use of Snakemake-INSaFLU, rather than delving deeply into data analysis. Therefore, the analysis I've conducted suggests interesting directions that I believe should be explored further in future work.

6

Future work

6.1 INSaFLU-Snakemake

6.1.1 Implementation

6.1.1.1 Sequencing Technology Identification

An important future improvement is the implementation of a sequencing technology identification step. This addition would involve analyzing the fastq data to automatically determine the sequencing technology used in its production. This advancement would pave the way for streamlined utilization of the same input files for both tools, particularly when handling sample descriptions containing essential metadata.

6.1.1.2 Output metadata and software versions

To get closer to the original implementation specific outputs should have the metadata of each sample, the software versions, and parameters. A better approach in INSaFLU-Snakemake would the use YAML files that define the software version of each environment created by the pipeline. With this information, the output would be always the correct version. INSaFLU uses hard-coded values which are prone to error, either by a mistype or forgetting to update the variable.

6.1.1.3 Modularization of Snakemake Files

A promising direction for enhancing the maintainability and scalability of the INSaFLU-Snakemake is the reorganization of Snakemake files (smk) into modular files. Currently, the focus is on rule files dedicated to a specific software. The alternative would be files that have all the rules of a specific module.

6.1.1.4 Transition to Python 3.11 and Beyond

A significant improvement in development would involve adopting Python 3.11 and newer versions exclusively. While earlier versions, such as Python 3.6, may have been needed by

software dependencies during initial development, a comprehensive transition to Python 3.11 offers substantial benefits. Notably, it enables the integration of type annotations, enhancing code clarity and helping in structural understanding. Embracing this change empowers developers to leverage static code analysis tools for further improvement.

6.1.1.5 What are the limiting steps

The efficiency of consensus alignment exhibits an inverse relationship with the number of samples and the size of the genome being analyzed. While the alignment process is notably swift with a smaller sample set, there is a discernible slowdown as the volume of samples escalates. A contrast in processing speed becomes evident when comparing the genomes of Monkeypox and SARS-CoV-2. Due to its genome size being 6.6 times more extensive, the alignment of the monkeypox genome is significantly slower for the same number of samples.

6.1.1.6 Optimized File Management

To streamline operations and reduce redundancy, it is imperative to address the current practice of copying files that undergo analysis by tools like snippy, iVar, or medaka into the project directory. Potential solutions include relocating these files into the project directory or processing them directly within the project folder. The biggest factor against that approach is the processing of the same samples with the same parameters by the same software. Currently, that is skipped and the files are directly copied into the project. Various options are available in the future but analyzing directly in the project folder would be my choice.

6.2 Extending INSaFLU-Snakemake

6.2.1 Another alternative to changes in the pipeline

The snakemake implementation of INSaFLU facilitates testing new software alternatives for the pipeline. In this context, iVar replaced snippy in the consensus generation and variant calling steps. While other steps in the pipeline have potential alternative options, Mafft stands out as a bottleneck that could benefit from an alternative. One possible alternative to Mafft could be NextAlign. However, it would be ideal to have software that provides deterministic output and listens to consensus generation step and coverage validation, progressively aligning the consensus as they are produced. Unfortunately, such software is currently unavailable. Another consideration is the conda incompatibilities with the Medaka version that the website uses. In this case, it is just a detail because the output is the same, but it still makes finding an alternative to Medaka compelling.

6.3 Enabling execution in a SLURM cluster

6.3.1 Exploring Other Platforms for Running INSaFLU

Snakemake is a robust WMS with broad community support and compatibility with various platforms. Not only is this WMS well-prepared for SLURM and generic clusters, as previously discussed, but it also extends support to platforms like Google Cloud Life Sciences, Google Cloud Engine with Kubernetes, TIBANNA in Amazon Web Services, Azure Batch, and GA4GH TES. Comprehensive guides on setting up and running Snakemake on these diverse platforms are provided in the Snakemake documentation. Although some platforms may require more steps to set up, the extra effort should prove worthwhile for users with large datasets.

6.4 Showcase INSaFLU-Snakemake by analyzing the Portuguese Monkeypox dataset

6.4.1 Exploring Genes in SNP-Enriched Regions and Genomic Features of Monkeypox

In future research with the Monkeypox dataset, it will potentially be valuable to investigate genes located within regions exhibiting a higher enrichment of SNPs. Additionally, exploring any distinctive genomic characteristics of Monkeypox that could provide insights into explaining this observation should be considered. These investigations may contribute to a deeper understanding of the Monkeypox genome and the factors influencing the observed pattern.

Bibliography

[1] J. M. Lourenço. *The NOVAthesis Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (cit. on p. i).

[2] D. Wu et al. "The SARS-CoV-2 outbreak: What we know". In: *International Journal of Infectious Diseases* 94 (2020-05), pp. 44–48. ISSN: 18783511. DOI: 10.1016/j.ijid.2020.03.004 (cit. on p. 1).

[3] B. Hu et al. "Characteristics of SARS-CoV-2 and COVID-19". In: *Nature Reviews Microbiology* 19 (3 2021-03), pp. 141–154. ISSN: 1740-1526. DOI: 10.1038/s41579-020-00459-7 (cit. on p. 1).

[4] Y. Huang, L. Mu, and W. Wang. "Monkeypox: epidemiology, pathogenesis, treatment and prevention". In: *Signal Transduction and Targeted Therapy* 7 (1 2022-12). ISSN: 20593635. DOI: 10.1038/s41392-022-01215-4 (cit. on p. 1).

[5] E. C. for Disease Prevention and Control. *ECDC. Epidemiological update: monkeypox multi-country outbreak.* Tech. rep. European Centre for Disease Prevention and Control, 2022 (cit. on p. 1).

[6] D. Buckeridge and G. Cadieux. "Surveillance for Newly Emerging Viruses". In: *Perspectives in Medical Virology* 16 (2006), pp. 325–343. ISSN: 01687069. DOI: 10.1016/S0168-7069(06)16013-9 (cit. on p. 1).

[7] S. Datta. "Next-generation sequencing in clinical virology: Discovery of new viruses". In: *World Journal of Virology* 4 (3 2015), p. 265. ISSN: 2220-3249. DOI: 10.5501/wjv.v4.i3.265 (cit. on p. 1).

[8] S. Goodwin, J. D. McPherson, and W. R. McCombie. "Coming of age: ten years of next-generation sequencing technologies". In: *Nature Reviews Genetics* 17 (6 2016-06), pp. 333–351. ISSN: 1471-0056. DOI: 10.1038/nrg.2016.49 (cit. on p. 1).

[9] V. Borges et al. "INSaFLU: An automated open web-based bioinformatics suite 'from-reads' for influenza whole-genome-sequencing-based surveillance". In: *Genome Medicine* 10 (1 2018-06). ISSN: 1756994X. DOI: 10.1186/s13073-018-0555-0 (cit. on p. 1).

[10] "FASTQC". In: *https://www.bioinformatics.babraham.ac.uk/projects/fastqc/* () (cit. on p. 2).

[11] A. M. Bolger, M. Lohse, and B. Usadel. "Trimmomatic: A flexible trimmer for Illumina sequence data". In: *Bioinformatics* 30 (15 2014-08), pp. 2114–2120. ISSN: 14602059. DOI: 10.1093/bioinformatics/btu170 (cit. on p. 2).

[12] W. D. Coster et al. "NanoPack: visualizing and processing long-read sequencing data". In: *Bioinformatics* 34 (15 2018-08), pp. 2666–2669. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bty149 (cit. on p. 2).

[13] "Abricate". In: *https://github.com/tseemann/abricate* () (cit. on p. 2).

[14] A. Bankevich et al. "SPAdes: A new genome assembly algorithm and its applications to single-cell sequencing". In: *Journal of Computational Biology* 19 (5 2012-05), pp. 455–477. ISSN: 10665277. DOI: 10.1089/cmb.2012.0021 (cit. on p. 2).

[15] "https://github.com/monsanto-pinheiro/getCoverage". In: () (cit. on p. 2).

[16] T. Seemann. *Snippy*. https://github.com/tseemann/snippy. 2015 (cit. on p. 3).

[17] H. Li and R. Durbin. "Fast and accurate short read alignment with Burrows–Wheeler transform". In: *Bioinformatics* 25 (14 2009-07), pp. 1754–1760. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btp324 (cit. on p. 3).

[18] E. Garrison and G. Marth. "Haplotype-based variant detection from short-read sequencing". In: (2012-07). URL: http://arxiv.org/abs/1207.3907 (cit. on p. 3).

[19] P. D. Tommaso et al. "Nextflow enables reproducible computational workflows". In: *Nature Biotechnology* 35 (4 2017-04), pp. 316–319. ISSN: 1087-0156. DOI: 10.1038/nbt.3820 (cit. on pp. 3, 5).

[20] "https://github.com/nanoporetech/medaka". In: () (cit. on p. 3).

[21] H. Li. "Minimap2: pairwise alignment for nucleotide sequences". In: *Bioinformatics* 34.18 (2018-05), pp. 3094–3100. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bty191. eprint: https://academic.oup.com/bioinformatics/article-pdf/34/18/3094/48919122/bioinformatics_34_18_3094.pdf. URL: https://doi.org/10.1093/bioinformatics/bty191 (cit. on p. 3).

[22] P. Danecek et al. "Twelve years of SAMtools and BCFtools". In: *GigaScience* 10 (2 2021-01). ISSN: 2047-217X. DOI: 10.1093/gigascience/giab008 (cit. on p. 3).

[23] K. Katoh. "MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform". In: *Nucleic Acids Research* 30 (14 2002-07), pp. 3059–3066. ISSN: 13624962. DOI: 10.1093/nar/gkf436 (cit. on p. 3).

[24] M. N. Price, P. S. Dehal, and A. P. Arkin. "FastTree 2 – Approximately Maximum-Likelihood Trees for Large Alignments". In: *PLoS ONE* 5 (3 2010-03), e9490. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0009490 (cit. on p. 3).

[25] G. Yachdav et al. “MSAViewer: interactive JavaScript visualization of multiple sequence alignments”. In: *Bioinformatics* 32 (22 2016-11), pp. 3501–3503. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btw474 (cit. on p. 3).

[26] B. Ribeiro-Gonçalves et al. “PHYLOViZ Online: web-based tool for visualization, phylogenetic inference, analysis and sharing of minimum spanning trees”. In: *Nucleic Acids Research* 44 (W1 2016-07), W246–W251. ISSN: 0305-1048. DOI: 10.1093/nar/gkw359 (cit. on p. 3).

[27] M. D. Wilkinson et al. “The FAIR Guiding Principles for scientific data management and stewardship”. In: *Scientific Data* 3 (1 2016-03), p. 160018. ISSN: 2052-4463. DOI: 10.1038/sdata.2016.18 (cit. on p. 4).

[28] conda contributors. *conda: A system-level, binary package and environment manager running on all major operating systems and platforms.* URL: https://github.com/conda/conda (cit. on p. 5).

[29] D. Merkel. “Docker: Lightweight Linux Containers for Consistent Development and Deployment”. In: *Linux J.* 2014.239 (2014-03). ISSN: 1075-3583 (cit. on p. 5).

[30] J. Leipzig. “A review of bioinformatic pipeline frameworks”. In: *Briefings in Bioinformatics* 18 (3 2017-05), pp. 530–536. ISSN: 14774054. DOI: 10.1093/bib/bbw020 (cit. on p. 5).

[31] F. Mölder et al. “Sustainable data analysis with Snakemake”. In: *F1000Research* 10 (2021-01), p. 33. ISSN: 2046-1402. DOI: 10.12688/f1000research.29032.1 (cit. on p. 5).

[32] T. G. Community. “The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2022 update”. In: *Nucleic Acids Research* 50.W1 (2022-04), W345–W351. ISSN: 0305-1048. DOI: 10.1093/nar/gkac247. eprint: https://academic.oup.com/nar/article-pdf/50/W1/W345/45189566/gkac247.pdf. URL: https://doi.org/10.1093/nar/gkac247 (cit. on p. 5).

[33] F Mölder et al. “Sustainable data analysis with Snakemake [version 1; peer review: 1 approved, 1 approved with reservations]”. In: *F1000Research* 10.33 (2021). DOI: 10.12688/f1000research.29032.1 (cit. on p. 5).

[34] P. Amstutz et al. “Common Workflow Language”. In: *figshare.com* (2016) (cit. on p. 6).

[35] M. R. Crusoe et al. “Standardizing Computational Reuse and Portability with the Common Workflow Language”. In: *Communications of the ACM* 65 (6 2022-06), pp. 54–63. ISSN: 15577317. DOI: 10.1145/3486897 (cit. on p. 6).

[36] J. Köster and S. Rahmann. “Snakemake—a scalable bioinformatics workflow engine”. In: *Bioinformatics* 28 (19 2012-10), pp. 2520–2522. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/bts480 (cit. on p. 6).

[37] W. Huang et al. "ART: a next-generation sequencing read simulator". In: *Bioinformatics* 28.4 (2011-12), pp. 593–594. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btr708. eprint: https://academic.oup.com/bioinformatics/article-pdf/28/4/593/48879907/bioinformatics_28_4_593.pdf. URL: https://doi.org/10.1093/bioinformatics/btr708 (cit. on p. 16).

[38] N. F. G. Chen et al. "Development of an amplicon-based sequencing approach in response to the global emergence of mpox". In: *PLOS Biology* 21 (6 2023-06), e3002151. ISSN: 1545-7885. DOI: 10.1371/journal.pbio.3002151 (cit. on p. 19).

[39] N. F. G. Chen et al. "Development of an amplicon-based sequencing approach in response to the global emergence of mpox". In: *PLOS Biology* 21.6 (2023-06), pp. 1–22. DOI: 10.1371/journal.pbio.3002151. URL: https://doi.org/10.1371/journal.pbio.3002151 (cit. on p. 19).

[40] J. Isidro et al. "Phylogenomic characterization and signs of microevolution in the 2022 multi-country outbreak of monkeypox virus". In: *Nature Medicine* 28 (8 2022-08), pp. 1569–1572. ISSN: 1546170X. DOI: 10.1038/s41591-022-01907-y (cit. on p. 19).

[41] C. R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (2020-09), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2 (cit. on p. 20).

[42] V. Mixão et al. "ReporTree: a surveillance-oriented tool to strengthen the linkage between pathogen genetic clusters and epidemiological data". In: *Genome Medicine* 15 (1 2023-06), p. 43. ISSN: 1756-994X. DOI: 10.1186/s13073-023-01196-1 (cit. on p. 20).

[43] F. Wu et al. "A new coronavirus associated with human respiratory disease in China". In: *Nature* 579 (7798 2020-03), pp. 265–269. ISSN: 0028-0836. DOI: 10.1038/s41586-020-2008-3 (cit. on p. 22).

[44] S. Sadeghpour et al. "Human apobec3 variations and viral infection". In: *Viruses* 13 (7 2021-07). ISSN: 19994915. DOI: 10.3390/v13071366 (cit. on p. 44).

[45] A. Ng et al. "The vaccinia virus A41L protein is a soluble 30 kDa glycoprotein that affects virus virulence". In: *Journal of General Virology* 82 (9 2001-09), pp. 2095–2105. ISSN: 0022-1317. DOI: 10.1099/0022-1317-82-9-2095 (cit. on p. 55).

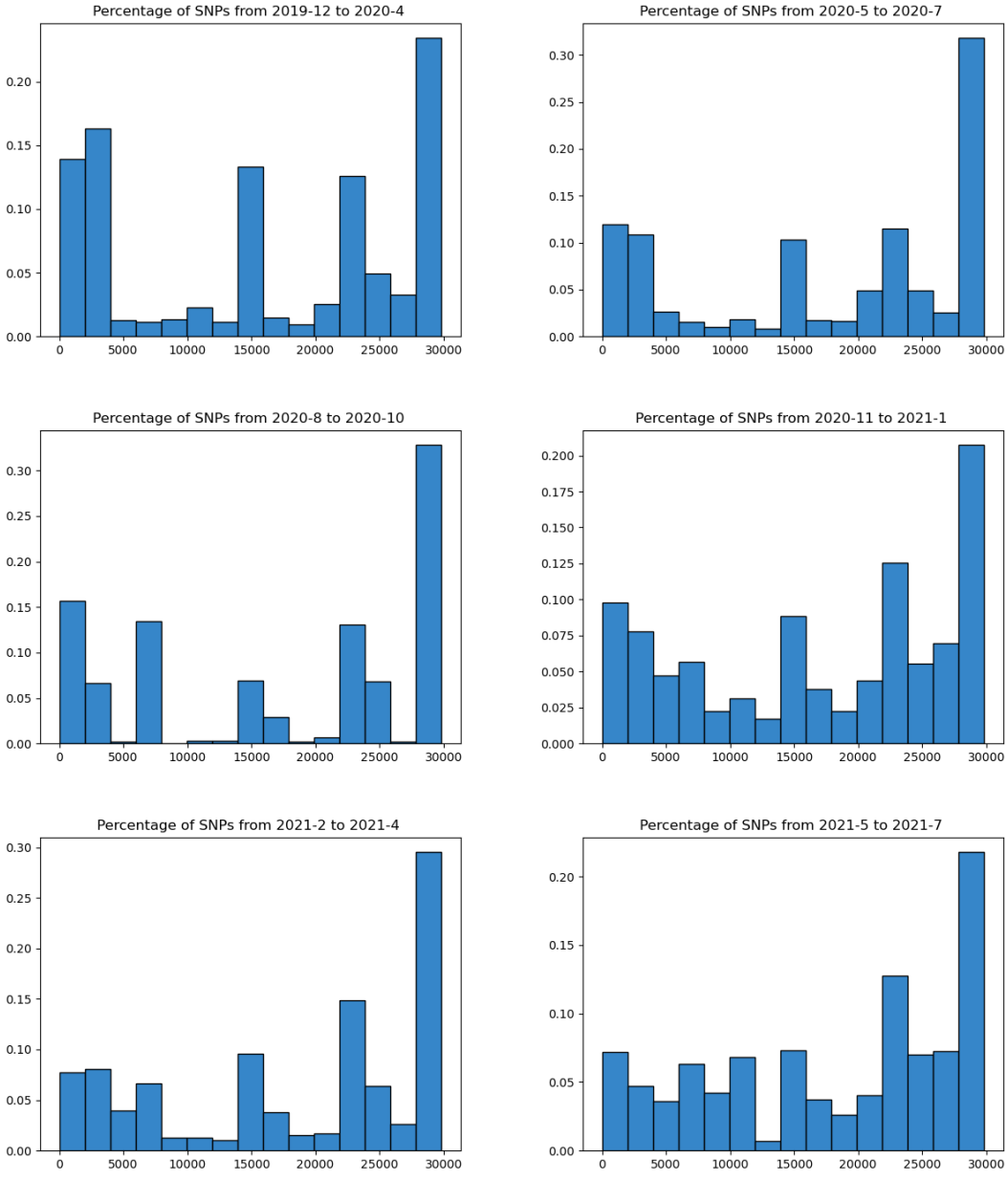

Figure .1: 2019-2021

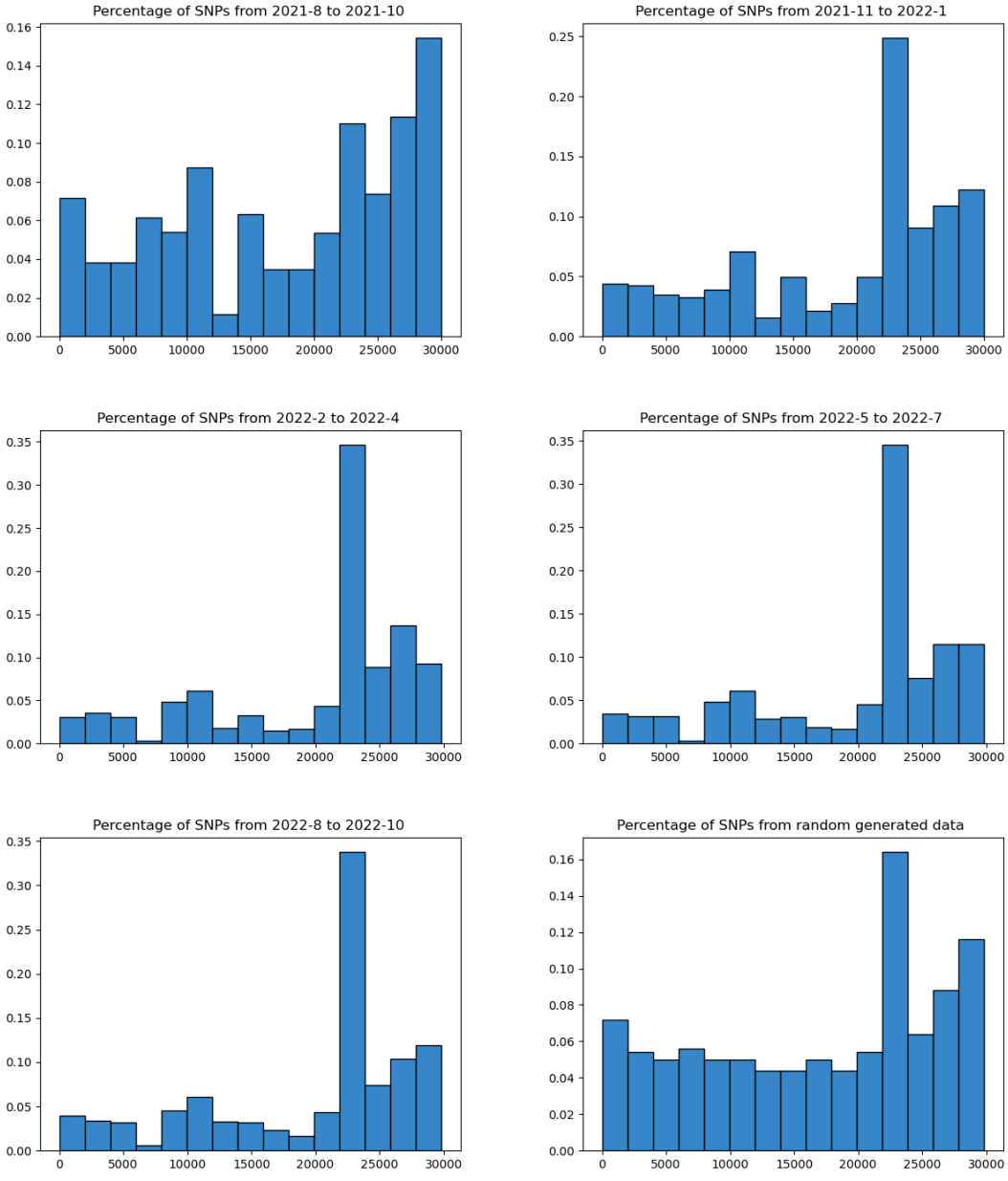

Figure .2: 2021-2022

Table .1: Software versions in INSaFLU-Snakemake.

Software	**Version**
snippy	3.2
trimmomatic	0.39
snpeff	5.1
mafft	7.313
nanostat	1.4.0
spades	3.11.1
nextalign	2.8.0
freebayes	1.1.0
pangolin	4.2
tabix	1.11
fasttree	2.1.11
bcftools	1.9
fastqc	0.11.9
abricate	0.8
nanofilt	2.6

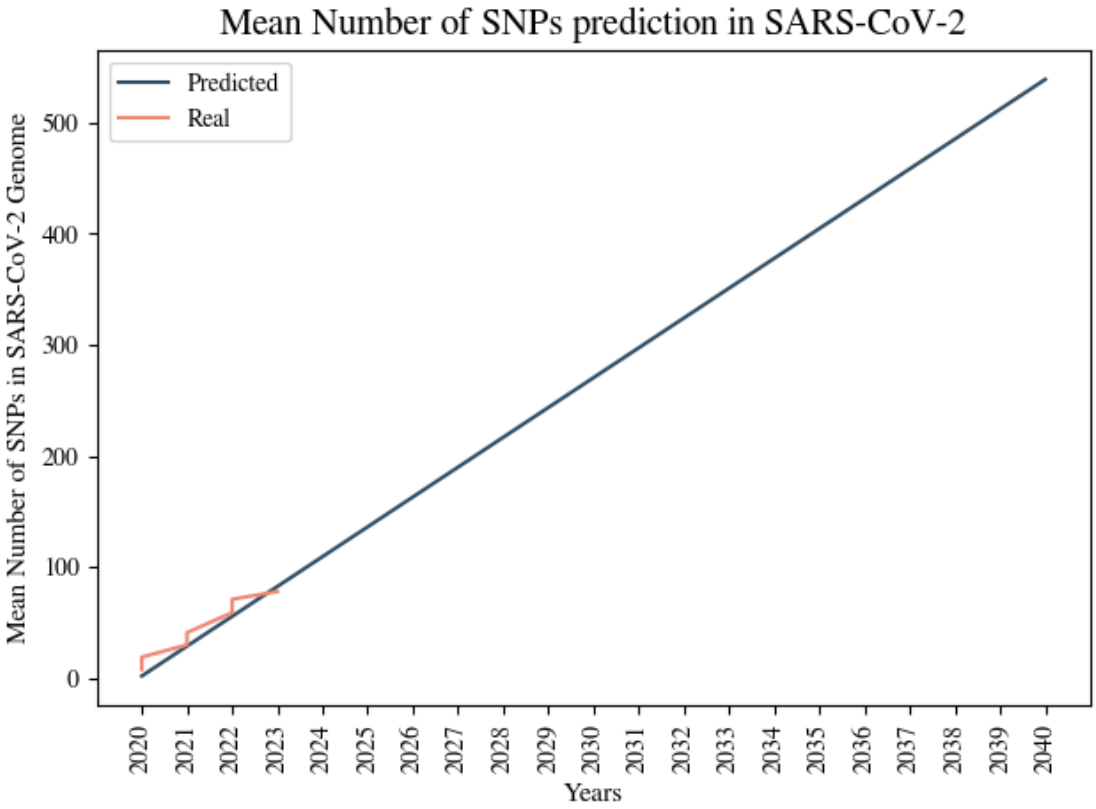

Figure .3: Prediction from 2020 to 2040 of the mean number of SNPs in SARS-CoV-2.

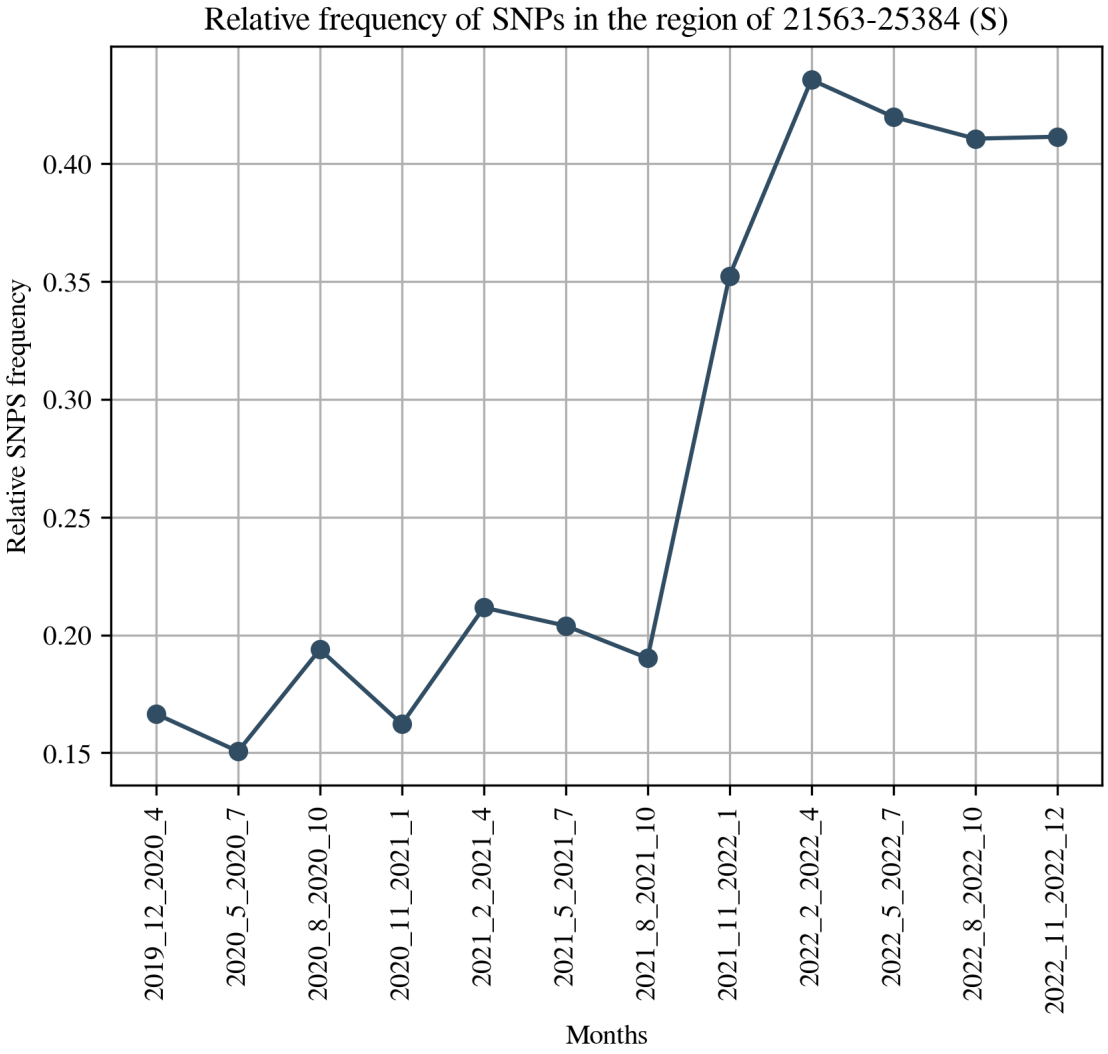

Figure .4: Relative percentage of SNPs in the Spike gene positions.

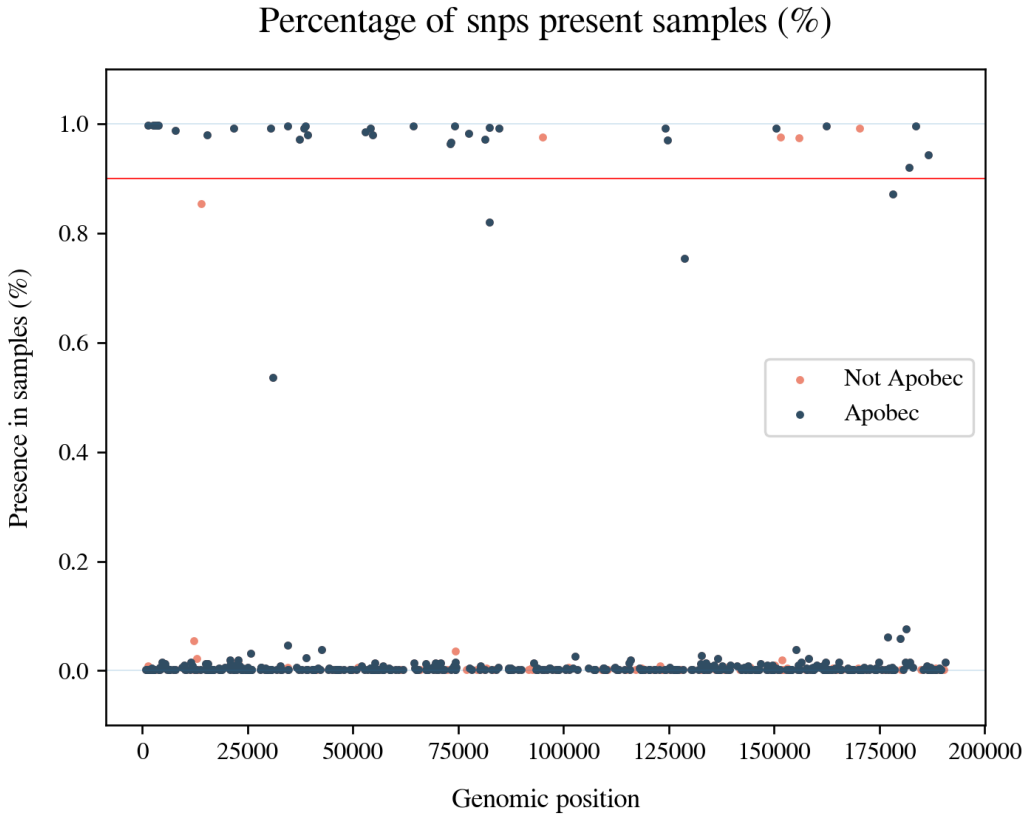

Figure .5: Plot with SNPs presence in a percentage of samples and their position in the Genomic Position using iVar. Color-coded according to their annotation, APOBEC in blue and non-APOBEC in pink.

UNIVERSIDADE NOVA
DE LISBOA