NextFlow pipeline for SARS-CoV-2 Illumina data

Table of contents

Quickstart

Installation and usage

1. Install Docker (https://docs.docker.com/desktop/install/linux-install/)

2. Install NextFlow

```
curl -s https://get.nextflow.io | bash
mv nextflow ~/bin
```

3. Clone the repository:

```
git clone https://github.com/mkadlof/nf_illumina_sars
```

4. Copy `third-party/modeller/config.py.template` to `third-party/modeller/config.py` and replace the line

`license = 'YOUR_MODELLER_KEY'`

with the actual Modeller key you own. If you don't have one, you can get a free academic license here (https://salilab.org/modeller/registration.html).

5. Build three containers:

```
docker build --target production -f Dockerfile-main -t
nf_illumina_sars-3.0-main .
docker build --target prodcution -f Dockerfile-manta -t
nf_illumina_sars-3.0-manta .
docker build --target updater -f Dockerfile-main -t nf_illumina_sars-
3.0-updater:latest .
```

6. Download latest version of external databases:

In project root dir run:

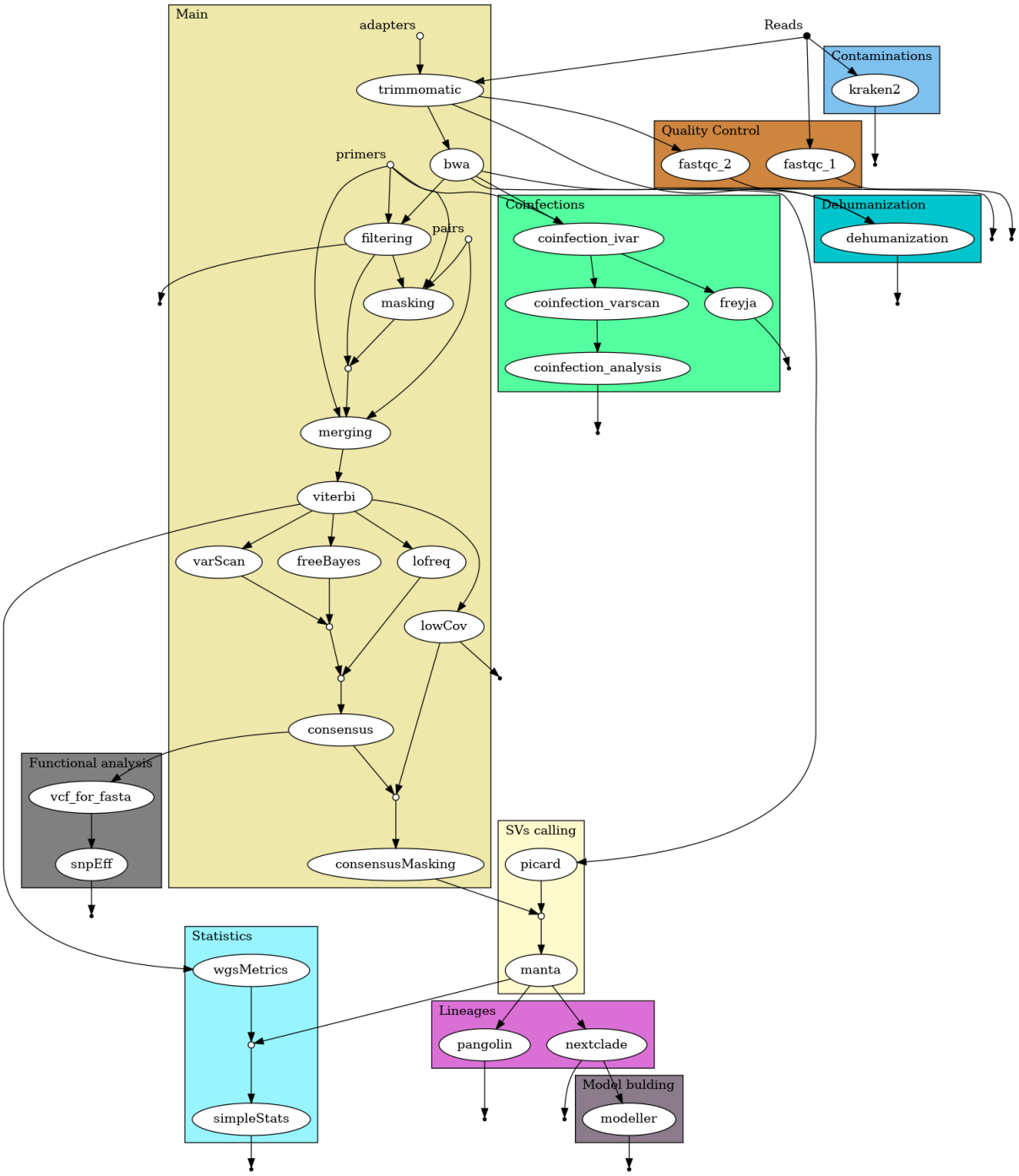
```
./update_external_databases.sh
```

This should fill directories in `data/pangolin` and `data/nextclade`. For more details read the chapter External databases updates.

7. Copy `run_nf_pipeline.sh.template` to `run_nf_pipeline.sh` and fill in the paths to the reads and output directory.
8. Run the pipeline:

```
./run_nf_pipeline.sh
```

Pipeline overview



Overview of the pipeline.

NextFlow pipeline for SARS-CoV-2 Illumina data consist of 30 NextFlow modules. The steps are depicted on image in Pipeline overview (`IndexGenome` module which gives input to many

other modules was hidden for clarity). Modules are grouped into logical sections, by their function.

Main section

Goal of **Main** section is to perform mapping with bwa (https://bio-bwa.sourceforge.net/) aligner, filtering, small indel calling by three callers (varScan (https://varscan.sourceforge.net/), freeBayes (https://github.com/freebayes/freebayes) and lofreq (https://csb5.github.io/lofreq/)), identify low quality / coverage regions and finally obtain consensus sequence.

Quality control

This section executes FastQC (https://www.bioinformatics.babraham.ac.uk/projects/fastqc/). Test is run on raw dataset and after trimming adapters with Trimmomatic (http://www.usadellab.org/cms/?page=trimmomatic).

Contamination detection

This section test if sample is nt contaminated with other species than SarsCov-2. Kraken2 (https://ccb.jhu.edu/software/kraken2/) is used.

Dehumanization

This section is used to create a subset of reads in FASTQ format without any reads from organisms other than SARS-CoV-2. Own python scripts are used.

Coinfections

This section is to detect infections with more than one variant of Sars-Cov-2 virus. Coinfections are detected with both own python scripts and Freya (https://andersen-lab.github.io/Freyja/index.html).

Functional analysis

This module returns the effects of nucleotide mutations onto protein sequence. We employ SnpEff (http://pcingola.github.io/SnpEff/).

SVs Calling

This module is capable to detect large genome rearrangements (*Structural variants*). This is done by down sampling reads with Picard (https://broadinstitute.github.io/picard/) and

running Manta (https://github.com/Illumina/manta).

Statistics

Some simple stats ara calculated with both - own python script and tools from Picard tools.

Lineages

After SVs calling pango (https://cov-lineages.org/resources/pangolin.html) line and nextclade (https://clades.nextstrain.org/) lineages are identified.

Building Spike protein model

As a last step Spike protein model is built with Modeller (https://salilab.org/modeller/)

Hardware requirements

Platform

Pipeline is intended to run on Unix-like computing server with x86_64 architecture.

Memory and CPU

Pipeline consist of multiple steps (processes) that are run in separate containers possibly concurrently. Each process has its own hardware requirements. Proceses vary highly in their demands, from very low to very high. Some may benefit from multiple cores while others are single-threaded or fast enough to not require more than one core. Exact requirements depend on expected number of samples analyzed in parallel.

During our tests we run the pipeline for 32 samples in parallel on a machine with 96 cores (Intel(R) Xeon(R) Gold 6240R CPU @ 2.40GHz) and 503 GiB of RAM, and we faced out-of-memory issues. We introduced limits on memory-intensive processes (BWA and Kraken2) to maximum 5 concurrent instances, to avoid OOM killer.

Single sample mode

In case of running the pipeline in single sample mode we recommend using at least 16 cores and 64 GiB of RAM.

Multiple samples mode

In case of running set of samples in parallel we recommended using 4 cores and ~100 GiB of RAM per sample.

Storage requirements

Total storage requirements is **~60 GiB** of constant data, and further **~3.3 GiB per sample**.

Performance

Many processes in the perform a lot of I/O operations, thus pipeline definitely can benefit from fast storage. We recommend store external databases, and temporary files in fast storage like NVMe SSD in RAID 0. It also may be beneficial to store it on in-memory filesystem like `tmpfs`, however no extensive tests were performed.

Docker images sizes:

Pipeline consist of three docker images two for computations and one is wrapper for external databases updates.

Image	Size
`nf_illumina_sars-3.0-main`	1.82 GiB
`nf_illumina_sars-3.0-updater`	257 MiB
`nf_illumina_sars-3.0-manta`	1.26 GiB
Total	**3.33 GiB**

Databases sizes:

Pipeline require access to external databases. Total size of databases is ~**56 GiB**.

Database	Size
pangolin	~90 MiB
nextclade	~1.3 MiB
kraken	~55 GiB
freyja	~100 MiB
Total	**~56 GiB**

Temporary files and results

Pipeline generates a lot of temporary files, which are stored in `work` directory. According to our tests on EQA2023 dataset (32 samples) took ~105 GiB of disk space ~**3.3 GiB per sample**. Please note that those tests are not representative, and real life data may vary significantly.

GPU requirements

Pipeline does not exploit GPU acceleration, so no GPU is required.

Software requirements

This pipeline is designed to run on a Unix-like operating system. The pipeline is written in Nextflow (https://www.nextflow.io/docs/latest/index.html), which is a language for writing bioinformatics pipelines. It is designed to be portable and scalable, and can be run on a variety of platforms, including local machines, clusters, and cloud computing environments.

Our pipeline is containerized using Docker (https://www.docker.com/), which is a platform for developing, shipping, and running applications in containers. Containers allow a developer to package up an application with all the parts it needs, such as libraries and other dependencies, and ship it all out as one package. This makes it easy to deploy the application on any machine that supports Docker.

However, our containers are run differently than usual docker workflow. Containers are run by nextflow, instead of manual execution of docker. Nextflow take care of mounting volumes and deciding which container should be run and when.

Compatibility

Pipeline was tested with following software versions:

- **Operating system**:
  - Ubuntu 20.04.06 LTS
  - Debian 12.5
- **Docker**:
  - 24.0.7
  - 26.0.2
- **Nextflow**:
  - 23.10.1.5891

It is known that pipeline will not work with docker 20.10.5.

External databases updates

Some components of the pipeline require access to their two databases, which are updated roughly once every two weeks. Different software pieces need to be updated in different ways. To make this process as smooth and painless as possible, we prepared a dedicated Docker container exactly for this task, along with a bash script for running it with appropriate volume mounts. The script should be placed in either the cron or systemd timer and run on a weekly basis.

Updates procedure

Build the dedicated container:

```
docker build --target updater -f Dockerfile-main -t nf_illumina_sars-3.0-updater:latest .
```

Run the updater script. The working dir must be in project root directory.

```
update_external_databases.sh nextclade
update_external_databases.sh pangolin
update_external_databases.sh kraken
update_external_databases.sh freyja
```

Total size of downloads is ~55 GiB.

Database	Size
pangolin	~90 MiB
nextclade	~1.3 MiB
kraken	~55 GiB
freyja	~100 MiB

If everything work fine in directories `data\pangolin` and `data\nextclade` you should see downloaded content like below:

```
data/nextclade/
└── sars-cov-2.zip

data/pangolin/
├── bin
├── pangolin_data
└── pangolin_data-1.25.1.dist-info

data/kraken
└── k2_standard_20240112.tar.gz

data/freyja/
├── curated_lineages.json
├── lineages.yml
└── usher_barcodes.csv
```

It is recommended to put the following in crontab or equivalently systemd timer.

```
0 3 * * 6 cd /path/to/sars-illumina &&
bin/update_external_databases.sh nextclade
5 3 * * 6 cd /path/to/sars-illumina &&
bin/update_external_databases.sh pangolin
10 3 * * 6 cd /path/to/sars-illumina &&
bin/update_external_databases.sh freyja
15 3 1 */3 * cd /path/to/sars-illumina &&
bin/update_external_databases.sh kraken
```

Updates internals

The following section contain information what and how is updated. Unless you need to debug or refactor the code, and you followed guides in chapter "Updates procedure" in "External databases updates" you can safely skip it.

List of components that require updates

- Nextclade
- Pangolin

- Kraken
- Freyja

Updateing Nextclade database

Nextclade (https://docs.nextstrain.org/projects/nextclade/) is software for assigning evolutionary lineage to SARS-Cov2. To make it work properly, it requires a database which is updated roughly once every two weeks.

The recommended way of downloading dataset is using `nxtclade` tool.

```
nextclade dataset get --name sars-cov-2 --output-zip sars-cov-2.zip
```

Detailed manual is available https://docs.nextstrain.org/projects/nextclade/en/stable/user/datasets.html. Nextclade is downloading `index.json` from site: https://data.clades.nextstrain.org/v3/index.json, and based on that files it decide what to download and from where. Probably the same data are available directly on GitHub: https://github.com/nextstrain/nextclade_data/tree/master/data/nextstrain/sars-cov-2/wuhan-hu-1/orfs.

The command above will download a `sars-cov-2.zip` file in desired destination (default: `data/nextclade`). That directory have to be mounted inside `main` container. It is done by Nextflow in the `modules/nextclade.nf` module.

```
process nextclade {
    (...)
    containerOptions "--volume
${params.nextclade_db_absolute_path_on_host}:/home/SARS-
CoV2/nextclade_db"
    (...)
```

Updateing Pangolin database

Pangolin (https://github.com/cov-lineages/pangolin) (**P**hylogenetic **A**ssignment of **N**amed **G**lobal **O**utbreak **LIN**eages) is alternative to Nextclade software for assigning evolutionary lineage to SARS-Cov2.

To make it work properly, it requires a database that is stored in the Git repository pangolin-data (https://github.com/cov-lineages/pangolin-data).

Pangolin-data is actually a regular python package. Normal update procedure is via command: `pangolin --update-data`. It also can be installed by `pip` command. Keeping it inside main container is slightly tricky. We don't want to rebuild entire container just to update the database. We also don't want to keep the database inside the container, because it would force us to run the update before every pipeline run, which is stupid. The best solution is to mount the database from the host.

To achieve this goal we install the package externally to the container in designated path using host native `pip`.

```
pip install \
    --target data/pangolin \
    --upgrade \
    git+https://github.com/cov-lineages/pangolin-data.git@v1.25.1
```

> Make sure you entered proper version in the end of git url. The version number is also git tag. List of available tags with their release dates is here (https://github.com/cov-lineages/pangolin-data/tags).

Then that dir is mounted as docker volume inside the container (which is done automagically in the Nextflow module file):

```
process variantIdentification {
    containerOptions "--volume
${params.pangolin_db_absolute_path_on_host}:/home/SARS-CoV2/pangolin"
    (...)
```

During container build the `$PYTHONPATH` environment variable is set to indicate proper dir.

```
(...)
ENV PYTHONPATH="/home/SARS-CoV2/pangolin"
(...)
```

So the manual download consist of two steps:

1. Install the desired version of `pangolin-data` package in `data/pangolin` directory.
2. Provide absolute path to that dir during starting pipeline

   `--pangolin_db_absolute_path_on_host /absolute/path/to/data/pangolin`

Updateing Kraken database

Kraken 2 (https://ccb.jhu.edu/software/kraken2/) is a taxonomic classification system using exact k-mer matches. The pipeline utilizes it to detect contamination in samples. It requires a database of approximately 55 GiB, which could potentially be reduced to 16 GiB or 8 GiB, albeit at the cost of sensitivity and accuracy. It updated roughly quarterly. Skipping updates may result in skipping newer taxa.

Database may be built for user own (not recommended) or be downloaded from aws s3. DB is maintained by Kraken 2 maintainers. Here you can find more here (https://github.com/BenLangmead/aws-indexes), and here (https://benlangmead.github.io/aws-indexes/).

Downloading is via aws cli (`apt install awscli`), python library `boto3` or HTTP protocol. There are several types of databases, that differ with set of organism. We use and recommend using `standard` db, which is quite complete. There is also `nt` db which contain all RefSeq and GenBank sequences, but it's size and processing time is too high for routine surveillance.

Links for http downloads are available here (https://benlangmead.github.io/aws-indexes/k2). They are in form of: `https://genome-idx.s3.amazonaws.com/kraken/k2_DBNAME_YYYMMDD.tar.gz`

where DB name is one of following: `standard`, `standard_08gb`, `standard_16gb`, `viral`, `minusb`, `pluspf`, `pluspf_08gb`, `pluspf_16gb`, `pluspfp`, `pluspfp_08gb`, `pluspfp_16gb`, `nt`, `eupathdb48`.

Refer to official docs for more details.

In case of our pipeline we use python script that is using `boto3` library. It is part of `nf_illumina_sars-3.0-updater` container. For running this script simply pass `kraken` to the container during running.

We recommend using for this dedicated script: `update_externeal_databases.sh`.

Kraken DB will not be updated if local path already contain the file with the same name.

Updateing Freyja database

Freyja (https://andersen-lab.github.io/Freyja/index.html) is a tool to recover relative lineage abundances from mixed SARS-CoV-2 samples from a sequencing dataset.

Natively Freyja updates require installing Freyja python package (by default with conda) and running `freyja update` command. This command will download the latest version of curated lineages and usher barcodes. However, in our pipeline we do it simpler way. We download the files directly from the dedicated GitHub repository andersen-lab/Freyja-data (https://github.com/andersen-lab/Freyja-data) using regular `wget`, which is implemented in the `update_external_databases.sh` script, and `updater` container.

Running pipeline

Pipeline parameters

There are three types of flags that we use explicit, implicit and nextflow flags. Explicit MUST be provided during starting pipeline - they are paths for input files. Explicit parameters are mostly numeric values for various modules. They have set reasonable default values and usually there is no need to modify them. NextFlow flags aplay to the way how NextFlow is executed rather than to pipeline itself.

By convention pipeline params starts with two dash `--param_name`, while NextFlow flags starts with single dash `-param-name`.

Explicit pipeline parameters

```
./run_pipeline.sh \
--ref_genome 'path/to/reference/genome.fasta' \
--reads 'path/to/reads/sample_id_{1,2}.fastq.gz' \
--primers 'path/to/primers.bed' \
--pairs 'path/to/pairs.tsv' \
--adapters 'path/to/adapters.fa' \
--pangolin_db_absolute_path_on_host '/home/user/path/to/pangolin_db' \
--nextclade_db_absolute_path_on_host '/home/user/path/to/nextclade_db'
\
--kraken2_db_absolute_path_on_host '/home/user/path/to/kraken2_db' \
--freyja_db_absolute_path_on_host '/home/user/path/to/freyja'
```

`ref_genome` - path to reference genome fasta file

`reads` - path to reads in fastqc format. Must be gzipped and be in form: `sample_id_{1,2}.fastq.gz`. Name must be resorvable by shell into two different files. One for forward reads, and second fo reverse reads.

`primers` - primers in bed file format. Example is below.

```
MN908947.3  2826    2850    nCoV-2019_10_LEFT   1   +
TGAGAAGTGCTCTGCCTATACAGT
MN908947.3  3183    3210    nCoV-2019_10_RIGHT  1   -
```

```
TCATCTAACCAATCTTCTTCTTGCTCT
(...)
```

Common primers sets are included in `data/generic/primers` directory and include following:

`SARS1_partmerge_exp`, `SARS2_partmerge_exp`, `V1`, `V2`, `V3`, `V4`, `V4.1`, `V1200`, `V1201`

`pairs` - definition of primers identifiers in two column tab separated file. This file is included in corresponding every primers set in `data/generic/primers`. Structure of primer identifier is meaningful. Must match regexp `nCoV-2019_[1,2]_(LEFT,RIGHT)`. Example:

```
nCoV-2019_1_LEFT    nCoV-2019_1_RIGHT
nCoV-2019_2_LEFT    nCoV-2019_2_RIGHT
(...)
```

`adapters` - path to fasta file with adapters. Common adapters are included in `data/generic/primers`. Example:

```
>PrefixPE/1
TACACTCTTTCCCTACACGACGCTCTTCCGATCT
>PrefixPE/2
GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT
```

Implicit pipeline parameters

All implicit parameters are listed in main pipeline file `nf_pipeline.nf` with their reasonable defaults.

```
params.threads = 5
params.memory = 2024
params.quality_initial = 5
params.length = 90
params.max_number_for_SV = 200000
params.max_depth = 6000
params.min_cov = 20
params.mask = 20
params.quality_snp = 15
params.pval = 0.05
params.lower_ambig = 0.45
```

```
params.upper_ambig = 0.55
params.ref_genome_id = "MN908947.3"
```

To run pipeline with modified parameter simply add appropriate flag:

```
./run_nf_pipeline.sh --threads 10
```

`--threads`, positive integer, number of threads used by couple of different modules. In single sample mode should be set to 1/3 of available CPUS. In multisample mode should be adjusted empirically. Recommended value for decent server: 5.

`--memory`, positive integer in MiB, similar to above. Default: 2024

`--quality_initial`, positive integer in PHRED scale, per base quality threshold used in various filtering and reporting modules. Default: 5.

`--length`, positive integer - number of base pairs, minimum length of a read. Default: 90

`--max_number_for_SV` - positive integer, maximum number of reads in bam file for manta module, down sampled by Picard, Default: 200000

`--max_depth` - positive integer, number of base pairs, threshold for short indel callers, reads above this value will be discarded. This is used for speedup indel calling. Default: 6000

`--min_cov` - positive integer, number of base pairs, threshold below which mutation will not be called. Default: 20

`--mask` - positive integer, number of base pairs, below this coverage value, genome will be masked with N. Should be the same as `min_cov`. Default: 20

`--quality_snp` - positive integer, PHRED scale, minimum quality of a base for INDEL calling, Default: 15

`--pval` - float from range [0; 1], minimal probability for INDEL calling, Default: 0.05

`--lower_ambig` and `--upper_ambig` - float from range [0, 1], if fraction of reads introducing alternative allel, fall within this range the position will be classified as *ambiguity*. `upper_ambig` must be greater than `lower_ambig`. Default: [0.45; 0,55]

`--ref_genome_id` - string, identifier of reference genome. Do not change unless you know what you are doing. Default: `MN908947.3`

Nextflow parameters

This parameters comes with Nextflow and should not be modified without solid reason.

`-config` path to nextflow config file. Default file `nextflow.config` is provided with repo.

`-with-report` path to report from pipeline execution. May be safely disabled. Default: `report.html`

`-with-dag` path tu file with pipeline graph. May be safely disabled Default: `flowchart-raw.png`

`-with-docker` Docker image used for execution processes. Strictly required. Default: `nf_illumina_sars-3.0-main:latest`

`-resume` Control if restarted pipeline should use cached results or not. Irrelevant in production environment, since every sample will be always run exactly once. In development or during debug may significantly speed up things.

Pipeline Steps

Main section

- trimmomatic - removing adapters
- bwa - mapping reads
- filtering - filtering bad quality reads
- masking - masking primers
- merging - intermediate step
- viterbi - improving alignment
- lowCov - detecting low coverage regions
- varScan - small INDEL caller
- freeBayes - small INDEL caller
- lofreq - small INDEL caller
- consensus - consensus sequence of three callers
- consensusMasking - masking low coverage regions

Quality Check section

- fastqc_1 - check raw reads
- fastqc_2 - check reads after trimmomatic

Contaminations section

- Kraken2 - detecting other reads from other organisms

Coinfections section

- coinfections_ivar - intermediate step
- coinfections_varscan - intermediate step
- coinfection_analysis - detecting coinfections
- freyja - detecting coinfections - alternative method

Dehumanization section

- dehumanization - removing non viral reads

Functional analysis

- vcfForFasta - intermediate step
- snpEff - detecting effects of mutations on protein sequence

SV Calling section

- picard - intermediate step
- manta - detecting large SVs

Lineages section

- pangolin - detecting pango line
- nextclade - detecting nextclade line

Model building section

- modeller - building Spike protein model

IndexGenome

```
samtools faidx ${reference_fasta}
```

Many modules require a reference genome as input (either a standalone FASTA file or with a FAI index). This module creates an index once and passes it to the required modules. For clarity in the pipeline visualization diagram, this module is hidden.

Quality Control: Modules fastqc 1 and 2

```
fastqc --format fastq \
       --threads ${params.threads} \
       --memory ${params.memory} \
       --extract \
       --delete \
       --outdir . \
       ${reads[0]} ${reads[1]}
r1=\$(basename ${reads[0]} .fastq.gz)
r2=\$(basename ${reads[1]} .fastq.gz)
parse_fastqc_output.py \${r1}_fastqc/fastqc_data.txt
${params.quality_initial} >> ${prefix}_forward_fastqc.txt
parse_fastqc_output.py \${r2}_fastqc/fastqc_data.txt
${params.quality_initial} >> ${prefix}_reverse_fastqc.txt
```

FastQC modules are used for general sequencing quality assessment and detection of common sample issues. The module is run twice, first for the raw reads provided by the user, and then after the action of the trimmomatic program. The result is parsed by a custom Python script, and the results are returned to the users, allowing them to independently assess the sample.

Main: Module trimmomatic

```
java -jar /opt/trimmomatic/trimmomatic.jar PE ${reads[0]} ${reads[1]}
\
                                                forward_paired.fastq.gz \
                                                forward_unpaired.fastq.gz \
                                                reverse_paired.fastq.gz \
                                                reverse_unpaired.fastq.gz \

ILLUMINACLIP:${adapters}:2:30:10:8:True \

LEADING:${params.quality_initial} \

TRAILING:${params.quality_initial} \
                                                SLIDINGWINDOW:4:4 \
                                                MINLEN:${params.length}
```

At this stage, we remove nucleotides of low quality from the 5' and 3' ends of the reads as well as adapter sequences from the reads. Then, we filter out reads that do not meet the length criterion. Those pairs in which both reads meet the length criterion are saved to appropriate files named "paired", and those pairs in which one of the reads does not meet the length criterion after filtering are saved to files named "unpaired". Adapter sequences are provided in the form of .fasta files. In the pipeline, we use adapter sequences provided by the authors of the Trimmomatic program (e.g., default sequences from the TruSeq3-PE-2.fa file), which have been placed in the container in the /SARS-CoV2/adapters/ directory.

When setting the quality threshold, it is recommended to use low qualities (default is just 5), which may seem "unorthodox" at first glance. However, it should be remembered that we are dealing with amplicon-based sequencing, and the crucial information here is from which amplicon the read originates and whether it maps to any of the primers. Removing a fragment of the read from its 5' and 3' ends may lead to its incorrect classification, which will affect further analysis. On the other hand, not removing reads of very poor quality may result in incorrect read alignment. Below is an example of what excessive zeal in removing nucleotides from the 5'/3' end can lead to.

a. Original situation (X - any nucleotide; P – primer position in the reference genome; M - masked position in the read, i.e., not used for variant identification or coverage counting)

```
Read                          XXXXXXXXXXXXXXXXXXXXX
Reference genome XXXXXXXXXXXXXPPPPPPPPPXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

b. Mapping after rigorous removal of nucleotides from the 5’/3’ end of the read.

```
Read                                XXXXXXXXXXXXXXXXX
Reference genome XXXXXXXXXXXXXPPPPPPPPPXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

c. Final effect after masking primers for the read after rigorous removal of the 5’/3’ end (described in Step 5)

```
Read                                  MMMMXXXXXXXXXXX
Reference genome XXXXXXXXXXXXXPPPPPPPPPXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

In the above situation, it can be seen that the read does not originate from amplification using the shown primer (it maps before its start). However, after removing the initial nucleotides (which tend to have lower quality), we will assume that the read was indeed generated using this primer. Consequently, the entire read region that maps to this primer will be masked, and we will not use this information in further analysis. Although this problem may seem insignificant due to excessive sequencing of SARS-CoV-2 samples with coverages exceeding tens of thousands, it has serious consequences in regions where the use of a given amplicon is small and the information conveyed by individual reads is very valuable. Additionally, please note that the variable $length is set to 90. This is related to the analysis of one of the EQA test sequences, where short reads (length ~50) were artificially boosting coverages near the 5’ end of the genome.

Main: Module bwa

```
bwa index ${reference_fasta}
bwa mem -t ${params.threads} -T 30 ${reference_fasta} ${reads[0]}
${reads[1]} | \
    samtools view -@ ${params.threads} -Sb -f 3 -F 2048 - | \
    samtools sort -@ ${params.threads} -o mapped_reads.bam -
samtools index mapped_reads.bam
```

We map using the BWA program. For consistency of results, we no longer utilize the option to use a different aligner (such as Bowtie or Minimap2). After mapping, the reads are filtered. We keep only those read pairs in which both reads have been mapped to the reference genome and are proper pairs (-f 3), while simultaneously removing additional/alternative mappings for the reads (-F 2048). The reads are then sorted and indexed.

Main: Module filtering

This script is extremely important. It checks a received BAM file and performs quasi-undersampling. The parameters selected above can only be modified by modifying the Python script. The following actions are performed within this script:

1. Identification and saving for further analysis of paired reads located within an amplicon. A paired read is considered to be inside an amplicon if (I) the start of mapping of such a pair is at least at the position corresponding to the first nucleotide of the left primer of the amplicon, and (II) the end of mapping of such a pair is before the last nucleotide of the right primer of the amplicon. This means that for a pair of reads, it is not required for at least one of the reads to map to any of the primers. What is required is not exceeding the boundaries set by a particular amplicon. At the same time, an empirical criterion limiting the number of reads subjected to analysis is applied:
    1. The number of read pairs whose mapping start is between the position of the first nucleotide of the left primer of a given amplicon and the position **up to 150** nucleotides towards the 3' end cannot exceed **4000**.
    2. The number of reads from point a and reads whose mapping **starts at least 150 nucleotides towards the 3'** end counting from the first nucleotide of the left primer cannot exceed **8000**.
    3. The number of reads from points a and b, as well as reads whose mapping **starts at least 250 nucleotides towards the 3'** end counting from the first nucleotide of the left primer, cannot exceed **12000**. Reads that are inside an amplicon but are not selected for further analysis due to exceeding the aforementioned threshold values **are not** further analyzed. Reads that do not meet the criterion of belonging to a single amplicon may undergo further analysis only if low usage is detected for at least one of the amplicons in the analyzed sample. Empirically, usage is considered low if fewer than **100** paired reads are assigned to a given amplicon.
2. In the second step, reads that most likely belong to a single amplicon, and their source is not a fusion of two neighboring amplicons within the same pool, are removed. Such reads must meet one of two criteria:
    1. They start at least **10** nucleotides towards the **5' end** counting from the first nucleotide of the left primer, and end before the last nucleotide of the right primer of the amplicon. The amplicon covering such a read must have high usage.

2. They start at the position corresponding to the first nucleotide of the left primer of the low-usage amplicon and end no further than **10** nucleotides towards the **3' end** counting from the last nucleotide of the right primer corresponding to the neighboring amplicon. The amplicon covering such a read must have high usage.

3. In the last step, reads remaining after filtering from points 1 and 3 are analyzed for the possibility of originating from an insert that is a fusion of a low-coverage amplicon and a neighboring amplicon.

    1. If there is an amplicon towards the 5' end from the low-usage amplicon, we check if the read pair maps to the region defined by the left primer of the low-usage amplicon towards the 5' end and the right primer of the neighboring amplicon. If in such a pair 50% of the length of any of the read pairs maps to the low-usage amplicon, and 10% of the length of the other read pair maps to the low-usage amplicon, such a read is included in the analysis. If nucleotides from such a pair cover the left primer of the low-usage amplicon towards the 5' end from the low-usage amplicon and the right primer of the low-usage amplicon, such positions are MASKED using the ivar program. For this purpose, an ad hoc .bed file is created.

    2. If there is an amplicon towards the 3' end from the low-usage amplicon, we check if the read pair maps to the region defined by the left primer of the low-usage amplicon and the right primer of the neighboring amplicon towards the 5' end. If in such a pair 50% of the length of any of the read pairs maps to the low-usage amplicon, and 10% of the length of the other read pair maps to the low-usage amplicon, such a read is included in the analysis. If nucleotides from such a pair cover the left primer of the low-usage amplicon and the right primer of the neighboring amplicon towards the 5' end, such positions are MASKED using the ivar program. For this purpose, an ad hoc .bed file is created. Below are illustrated several scenarios. It is assumed that amplicon 1 has high usage (over 100 mapping read pairs), and amplicon 2 has low usage (below 100). Amplicon 1 is located between P1_L and P1_R; amplicon 2 is located between P2_L and P2_R. Both amplicons come from the same pool. In the schema, "X" denotes any nucleotide. The directionality of reads in the pair is not shown.

Positions of elements in the genome.

```
----P1_L----(amplicon1)----P1_R----P2_L------(amplicon2)------P2_R--
```

Hypothetical read pairs:

(Version 1)

```
---------------XXXXXXXXXXXXXXXXXXXXXXXX-------------------------
------------------------------XXXXXXXXXXXXXXXXXXXXXXXX------------
```

(Version 2)

```
----------XXXXXXXXXXXXXX-----------------------------------------
--------------------------------XXXXXXXXXXXXX--------------------
```

(Version 3)

```
----------XXXXXXXXXXXXXXXX---------------------------------------
-----------------------------------------------XXXXXXXXXXXXXXXXX-
```

(Version 4)

```
XXXXXXXXXXX-------------------------------------------------------
----------------------XXXXXXXXXXXX-------------------------------
```

Version 1.

The read pair is inside an insert formed by the fusion of amplicons 1 and 2. At least half of the nucleotides from read 2 of the pair cover amplicon 2, and at least 10% of the nucleotides from read 1 of the pair cover amplicon 2. Such a read is included in further analysis. The reads do not cover either P1_L or P2_R, so the read passes without masking.

Version 2.

The read pair is inside an insert formed by the fusion of amplicons 1 and 2. Less than 50% of the nucleotides from both read 1 and read 2 of the pair cover amplicon 2. Such a read is not analyzed further.

Version 3.

The read pair is inside an insert formed by amplicons 1 and 2. At least half of the nucleotides from read 2 of the pair cover amplicon 2, and at least 10% of the nucleotides from read 1 of the pair cover amplicon 2. Such a read is included in further analysis. Read 2 covers P2_R, so the 3’ end of this read (bolded) will be masked.

Version 4.

The read pair is not inside an insert formed by amplicons 1 and 2. The read is not analyzed.

Main: Module masking

```
length=`echo "${params.length} - 40" | bc -l`
ivar trim -i ${bam} \
          -b ${primers} \
          -m \${length} \
          -f ${pairs} \
          -q ${params.quality_initial} \
          -e \
          -p ivar_trimmed_all
```

Standard ivar usage. Here, we invoke it only on reads that map to a single amplicon (reads from Step 4 points 1 a,b,c). These reads are stored in a file named `reads_inneramplicon_sort.bam`. At this stage, we DO NOT use reads that come from amplicon fusions, as those are removed in the Python script call in Step 4.

The flags for ivar signify:

- `-b` path to the amplicon scheme file
- `-m` minimum read length after masking. Symbolically, it indicates that a read can be 40 bases shorter than the length selected during script invocation (the minimum length read in Step 2 covering the longest EQA test primer would have this length after primer masking)
- `-q` additional trimming of nucleotides based on quality. This option cannot be turned off, so to prevent ivar from removing nucleotides, we set the flag to the same value as that for Trimmomatic in Step 2
- `-e` Keep all reads in the output file, including those not mapping to any primer. By default, ivar retains only those reads (not pairs, but individual reads from pairs) that map to any primer. In our case, all we expect from the program is to mask the regions in reads that come from the primer among the pool of predefined read pairs. We want to further analyze reads that are within the amplicon but do not cover the primer.
- `-p` prefix, the program will generate a bam file with this name

Main: Module merging

```
samtools merge -o clean_sort_dedup_trimmed_sort_tmp.bam
${filtering_bam} ${ivar_bam}
samtools sort -@ ${params.threads} -o
clean_sort_dedup_trimmed_sort.bam
clean_sort_dedup_trimmed_sort_tmp.bam
samtools index clean_sort_dedup_trimmed_sort.bam
```

At this stage, we use Samtools to merge reads belonging to one amplicon with ivar-masked primers from files containing reads from amplicon fusions where one amplicon had low usage. These files also had masked primers.

Main: Module viterbi

```
lofreq viterbi --ref ${reference_fasta} \
               --out clean_sort_dedup_trimmed_sort_viterbi.bam \
               ${bam}

lofreq indelqual --ref ${reference_fasta} \
                 --out forvariants.bam \
                 --dindel clean_sort_dedup_trimmed_sort_viterbi.bam
samtools sort -@ ${params.threads} \
              -o forvariants.bam \
              forvariants.bam
samtools index forvariants.bam
```

This fragment is part of the procedure recommended for identifying reads using the Lofreq program following GATK recommendations. Reads are realigned and a quality score is introduced for indel quality, enabling their identification by the Lofreq program. The above procedure does not affect the results of Varscan or Freebayes.

Main: Module lowCov

```
pysam_quality_mask_final.py ${bam} 10 ${params.mask}
```

To identify regions with low coverage, we utilize the "count coverage" function available within the pysam package. This function has an advantage over a similar function in the `bedtools` package because it allows counting coverage while considering nucleotide quality. The Python script takes 3 arguments. The first is the bam file from which coverage is calculated, the second is the quality threshold, only nucleotides mapping to a region with at least this value are considered. The last argument is the coverage value, only positions in the genome with coverage below this value are returned. The output is a file named `quality_mask.bed`. In this file, each position with coverage below the threshold is returned as a separate entry. The file has standard 3 columns. The first is the chromosome name, the second is the start of the region (0-indexed), and the third column is the end of the region (0-indexed). Like any bed file, the regions are closed on the left and open on the right. An example content of the file is:

```
MN908947.3    0    1
MN908947.3    1    2
MN908947.3    2    3
MN908947.3    2    4
MN908947.3    100    200
```

```
cat quality_mask.bed | bedtools merge -d 2 | \
                       awk 'BEGIN {OFS = "\t"}; {if (\$3-\$2 > 3)
print \$1,\$2,\$3}' >> low_coverage.bed
```

In the next step, using `bedtools`, regions in the quality_mask.bed file are merged if the ranges they cover are separated by no more than 2 nucleotides. If, after merging, a low coverage region is at least 4 nucleotides long, it is saved to the `low_coverage.bed file`; otherwise, the region is ignored. This empirical criterion helps avoid situations where coverage in a region oscillates around the value given in the `$mask` argument, and the region is alternately masked and unmasked by very short segments. The example content of the quality_mask.bed file shown above, after this step, would look like this:

```
MN908947.3    0    4
```

This information is stored in the `low_coverage.bed` file.

```
bedtools maskfasta -fi ${reference_fasta} \
                   -bed low_coverage.bed \
                   -fo lowcoverage_masked.fa
```

The final step is to create a fasta file with the reference genome where positions with low coverage are masked with Ns. For this purpose, we use the appropriate function of the `bedtools` program. The fasta file with such a genome is named `lowcoverage_masked.fa`.

Main: Module varScan

Below is the procedure on how we use and parse the varscan program:

```
samtools mpileup --max-depth ${params.max_depth} \
            --fasta-ref ${reference_fasta} \
            --min-BQ ${params.quality_snp} \
            ${bam} >> ${bam}.mpileup
```

Creating the mpileup file required by the program. The option `-d` (note: properly functioning from `samtools` version 1.09 onwards) ignores further reads mapping to a given position if the coverage exceeds the assumed value.

```
varscan_qual=`echo "${params.quality_snp} - 1" | bc -l`
java -jar /opt/varscan/VarScan.v2.4.6.jar pileup2cns ${bam}.mpileup \
     --min-avg-qual \${varscan_qual} \
     --p-value ${params.pval} \
     --min-var-freq ${params.lower_ambig} \
     --min-coverage ${params.min_cov} \
     --variants \
     --min-reads2 0 > detected_variants_varscan.txt
```

Calling varscan, the above options signify:

- `--min-coverage` – minimum coverage at a given position required for the program to identify a variant.
- `--min-reads2` - minimum number of reads supporting the variant, set to 0.
- `--min-avg-qual` – minimum read quality at this position required to include the read in the analysis.
- `--p-value` – threshold p-value the variant must achieve to be reported.
- `--min-var-freq` – minimum percentage of reads with a non-reference allele required to report a variant.
- `--variants` – besides SNPs, the program should also identify short INDELs.`

```
parse_vcf_output_final.py detected_variants_varscan.txt
${params.upper_ambig} ${params.pval}
```

A parser converting the text file `detected_variants_varscan.txt` to a vcf file format. Compared to the txt file, the vcf file introduces the following changes:

1. Positions reported as deletions/insertions relative to the reference genome, e.g.,

   ```
   MN908947.3    22204    T    +GAGCCAGAA
   ```

   Containing symbols "+" or "-" are corrected to:

   ```
   MN908947.3    22204    .    T    TGAGCCAGAA
   ```

2. Heterozygous positions, e.g.,

   ```
   MN908947.3    21766    .    ACATGTC    A
   ```

3. Positions where the frequency of using the alternative allele is greater than the value provided in the variable $upper_ambig are converted from ambiguous to the alternative allele version. Varscan returns ambiguous positions even with a reference allele frequency of 0.75, e.g.,

   ```
   MN908947.    25324    C    M (where the frequency of allele A is
   60%)
   ```

   are corrected to

   ```
   MN908947.3    25324    .    C    A
   ```

4. The QUAL field, which does not have a clear representation in the varscan output, is changed to 30 in the vcf file.

```
bcftools norm --check-ref w \
                --rm-dup all \
                --fasta-ref ${reference_fasta}\
                 detected_variants_varscan.vcf.gz | \
```

```
                    bcftools norm --check-ref w \
                                  --multiallelics -indels \
                                  --fasta-ref ${reference_fasta} |
\
                                       bcftools filter \
                                               --include "QUAL >=
\${qual} && AF >= ${params.lower_ambig} && DP >= ${params.min_cov}" >
detected_variants_varscan_final.vcf
```

"Sorting" the vcf file using the norm function from the bcftools package. The `-c w` option causes the program to return a warning instead of an error if ambiguous positions are present in the file, and further process the vcf file. The "-d all" removes duplicates, the `-m -indels` option splits multiallelic positions into individual entries. Generally, vcf files show the richness of changes identified by programs. Sometimes overlapping changes occur, sometimes programs return unusual mutation notations, multiallelic positions, etc. To organize such a vcf file, we call the above command. Then we ensure that the resulting records still meet the quality and coverage criteria. In the case of varscan, this is not as important because when parsing its output, we set the quality to 1 or 30. Simple examples of changes identified in sample 1 from EQA for the freebayes program.

Before:

```
MN908947.3    22204    .    TGA    TGAGCCAGAAGA
```

After:

```
MN908947.3    22204    .    T    TGAGCCAGAA
```

Or splitting a complex mutation Before:

```
MN908947.3    25334    .    GAAG    CACG,CACC,GACC,GAAC
```

After, splitting the mutation into components

```
MN908947.3    25334    .    GAA    CAC
MN908947.3    25334    .    GAAG    CACC
MN908947.3    25336    .    AG    CC
MN908947.3    25337    .    G    C
```

```
cat ${reference_fasta} | bcftools consensus --samples -
detected_variants_varscan_final.vcf.gz > varscan.fa
```

The above command incorporates all mutations present in the vcf file into the reference genome. The `-s -` option means that we ignore the genotype for the sample (we do not have multiple samples in the vcf file), and we introduce all mutations present in the vcf file. The result is a fasta file format with the genome.

Main: Module freeBayes

```
freebayes --limit-coverage ${params.max_depth} \
          --min-coverage ${params.min_cov} \
          --min-mapping-quality 20 \
          --min-base-quality ${params.quality_snp} \
          --use-mapping-quality \
          --fasta-reference ${reference_fasta} \
          --ploidy 1 \
          ${bam} > detected_variants_freebayes.vcf
```

Calling the freebayes program. The options used are:

- `--limit-coverage ${max_depth}` – limit coverage at a position to this value when identifying variants. Please note that quasi-downsampling used in Step 4 for read filtering does not guarantee that a given position will not exceed the value specified in the max_depth variable.
- `--min-coverage ${min_cov}` – minimum coverage for a position to be considered as a variant.
- `-m 20` – alignment quality of a read for its nucleotides to be considered in variant analysis.
- `-q ${quality_SNP}` – minimum nucleotide quality at a position in a read to be considered in variant counting.
- `-p 1` – ploidy of the analyzed organism.
- `-f ${input_genome}` – path to the reference genome.
- `-j` – use mapping qualities when calculating variant likelihood.

```
cat detected_variants_freebayes.vcf | \
    bcftools norm --check-ref w \
                  --rm-dup all \
                  --fasta-ref ${reference_fasta} | \
                      bcftools norm --check-ref w \
                                    --multiallelics -indels \
```

```
                                        --fasta-ref ${reference_fasta} >
detected_variants_freebayes_fix.vcf
```

Normalization of the VCF file similar to varScan (Main: Module varScan).

```
bcftools filter --include "QUAL >= \${qual} & INFO/DP >=
${params.min_cov} & (SAF  + SAR)/(SRF + SRR + SAF + SAR) >
${params.upper_ambig} " \
         detected_variants_freebayes_fix.vcf >
detected_variants_freebayes_fix_high.vcf
```

Freebayes cannot identify ambiguous positions, and at the position corresponding to the alternative allele in the VCF file, it always introduces a symbol: `A`, `T`, `G`, or `C`. Introducing the allele ambiguous symbol at such a position must be done manually. In the first step, we save to the `detected_variants_freebayes_fix_high.vcf` file all mutations in which the frequency of using the alternative allele exceeds the value provided in the `${upper_ambig}` variable and whose quality is greater than 15 (meaning p-value is less than 0.03).

```
bcftools filter --include "QUAL >= \${qual} & INFO/DP >=
${params.min_cov}  & (SAF  + SAR)/(SRF + SRR + SAF + SAR) >=
${params.lower_ambig}  & (SAF  + SAR)/(SRF + SRR + SAF + SAR) <=
${params.upper_ambig} " \
         detected_variants_freebayes_fix.vcf > tmp_low.vcf
introduce_amb_2_vcf.py tmp_low.vcf \
       detected_variants_freebayes_fix_ambig.vcf
```

Next, we extract mutations with allele alternative frequency between the values specified in the variables `$lower_ambig` and `$upper_ambig`. For these positions, we apply a simple Python script where instead of introducing the alt allele symbol, we introduce the ambiguous nucleotide symbol.

```
bcftools concat detected_variants_freebayes_fix_high.vcf.gz \
               detected_variants_freebayes_fix_ambig.vcf.gz | \
                   bcftools sort --output-type z >
detected_variants_freebayes_final.vcf.gz
```

Main: Module lofreq

```
lofreq call-parallel --pp-threads ${params.threads} \
                     --ref ${reference_fasta} \
                     --max-depth ${params.max_depth} \
                     --min-cov ${params.min_cov} \
                     --call-indels \
                     --out detected_variants_lofreq.vcf \
                     ${bam}
```

- `--pp-threads` – number of CPU threads used for computations
- `-f` – path to the reference genome
- `--max-depth` – cutoff for the maximum coverage considered for each position
- `-C` – minimum coverage required for a position to be considered carrying a variant
- `--call-indels` – also identify INDELs

In the case of the lofreq program, we do not use parameters related to nucleotide quality in the read and mapping quality to the reference genome. The issue with counting correct coverage in such situations by the lofreq program is described in the following GitHub issue: https://github.com/CSB5/lofreq/issues/80. To calculate the proportions of the reference and alternative alleles, we use the DP4 field from the VCF file. It should be noted that it does not always sum up to the value in the DP field. For example, position 19,985 for Sample 10 from the EQA test. This is a region where the mutation is just behind a deletion, and moreover, the sequence being deleted is a repeated element in a palindrome. Additionally, this field is erroneous in the case of INDELs, so they continue to be analyzed if they meet the coverage, quality, and reference allele frequency criteria defined by the variable `$lower_ambig`.

Then, the steps are identical to those for the freebayes program. The output_detected_variants_lofreq.vcf file is split into two: one with mutations with high usage of the alternative allele, and the other with similar usage of the reference and alternative alleles. The files are appropriately filtered, merged, and used to identify the final list of mutations. Intermediate files leading to the creation of a fasta file with the sample genome are created as with the other programs.

Main: Module consensus

```
cat ${freebayes_fa} ${lofreq_fa} ${varscan_fa} > tmp_to_consensus.fa
mafft --auto --inputorder --quiet tmp_to_consensus.fa >
tmp_to_consensus_aln.fa
gen_consensus_seq_final.py tmp_to_consensus_aln.fa
```

Generating consensus from three programs utilizes the gap_consensus function from the Biopython package called by the `gen_consensus_seq_final.py` script. In this function, the threshold value is set to 0.6. This means that at a given position in the alignment, the consensus is considered to be the nucleotide that occurs in at least 60% of the sequences. If there is no nucleotide meeting this criterion, "X" is entered. The Biopython function understands deletions, and they are also recorded in the consensus sequence. Since we have three sequences at each position, we can get values of 0.33 (each sequence at this position has a different allele, which is possible when one program returns the reference allele, another program the alternative allele, and the third program ambiguous positions), 0.66 (2 out of 3 sequences share a common allele), or 1 (all sequences have the same allele). Thus, a threshold of 0.6 means that each position is a consensus of at least two, any two, programs. The resulting consensus sequence then undergoes the standard procedure of masking regions with low coverage.

Main: Module consensuMasking

```
cat lowcoverage_masked.fa consensus.fa >tmp_consensus.fa
mafft --auto --inputorder --quiet tmp_consensus.fa >
tmp_consensus_aln.fa
get_N.py tmp_consensus_aln.fa
mv output_consensus_masked.fa consensus_masked.fa
```

The above commands aim to combine information about low coverage regions obtained in Module lowCov (Main: Module lowCov) with information about mutations identified by the varScan (Main: Module varScan), FreeBayes (Main: Module freeBayes) and Lofreq (Main: Module lofreq) programs. To achieve this, we align the sequences present in the files `lowcoverage_masked.fa` and `consensus.fa`, ` using the MAFFT program. Then, a simple script combines information about low coverage regions and mutations according to a straightforward logic.

1. If a low coverage region aligns with a region where no deletion was identified, the final sequence at that position is also masked.
2. If the low coverage sequence is not masked at a given position, we enter positions as in the sequence generated by the varscan program.

Finally, from the resulting sequence, we remove the deletion symbol and obtain the final sample genome sequence, which is found in the file `consensus_masked.fa`.

Coinfections: Modules ivar, varscan, analysis

The co-infection section consists of a preparatory module whose task is to trim primer sequences from the sequences.

```
ivar trim -i ${mapped_reads} \
          -b ${primers} \
          -m ${params.length} \
          -q ${params.quality_initial} \
          -e \
          -p for_contamination
(...)
samtools mpileup --max-depth 10000 \
             --fasta-ref ${reference_fasta} \
             --min-BQ ${params.quality_snp} \
             for_contamination_sorted.bam >> for_contamination.mpileup
```

Next, the result is passed to the VarScan program, which identifies mutations in unfiltered samples.

```
varscan_qual=`echo "${params.quality_snp} - 1" | bc -l`
java -jar /opt/varscan/VarScan.v2.4.6.jar pileup2snp
${for_contamination_mpileup} \
                                                --min-avg-qual
\${varscan_qual} \
                                                --p-value 0.9 \
                                                --min-var-freq 0.05 \
                                                --min-coverage 20 \
                                                --variants \
                                                --min-reads2 0 >
detected_variants_varscan_contamination.txt
```

The final module compares allele frequency distributions with known samples that have been identified as co-infected (based on EQA23 tests) and ultimately answers the question about the probability of co-infection occurrence.

Coinfections: Module Freyja

```
mkdir variants_files depth_files demix_files
freyja variants mapped_reads.bam --variants
variants_files/test.variants.tsv --depths depth_files/test.depth --ref
${reference_fasta}
freyja demix variants_files/test.variants.tsv depth_files/test.depth -
-output demix_files/test.output --confirmedonly --barcodes
/home/external_databases/freyja/usher_barcodes.csv
freyja aggregate demix_files/ --output coinfections.tsv
```

Freyja is an alternative tool to detect coinfections in SARS-CoV-2 samples.

> ⚠ The method uses lineage-determining mutational “barcodes” derived from the UShER global phylogenetic tree as a basis set to solve the constrained (unit sum, non-negative) de-mixing problem.

Documentation (https://andersen-lab.github.io/Freyja/index.html)

Contaminations: Module Kraken2

```
kraken2 --db /home/external_databases/kraken2 \
        --report raport_kraken2.txt \
        --threads ${params.threads} \
        --gzip-compressed \
        --minimum-base-quality 30 \
        --use-names ${reads[0]} ${reads[1]} >>
raport_kraken2_individualreads.txt 2>&1
```

Kraken2 is a tool capable of detecting contamination in a sample by aligning reads to genomes in a database and assessing their taxonomic origin. The principle of Kraken involves mapping reads to genomes and performing read-based classification against a reference database.

As a result, users obtain information about the proportions of reads originating from different organisms.

Kraken requires a large database for classification. Details are described in updates.md#external-databases-updates (External databases updates).

Dehumanization: Module Dehumanization

```
samtools view mapped_reads.bam | cut -f1 | sort | uniq >>
lista_id_nohuman.txt
seqtk subseq ${reads[0]} lista_id_nohuman.txt >>
forward_paired_nohuman.fq
seqtk subseq ${reads[1]} lista_id_nohuman.txt >>
reverse_paired_nohuman.fq

gzip forward_paired_nohuman.fq
gzip reverse_paired_nohuman.fq
```

The goal of this module is to remove reads that do not map to the reference genome to avoid loading human reads into external databases. It creates a new set of purified ang gziped FASTQ files.

Functional analysis: vcfForFasta

```
prep_own_vcf.py ${reference_fasta} consensus.fa \$N \${vcf_input}
\${vcf_output}
```

A simple module designed to transform a fasta file containing a consensus sequence derived from three programs (varScan, FreeBayes, and Lofreq) into a VCF file format. The program has one parameter `-N`, which specifies the maximum gap size between mutations to merge them into a single complex mutation. This parameter is set within the module, and changing it requires rebuilding the container `nf_illumina_sars-3.0-main`.

Functional analysis: snpEff

```
java -jar /opt/snpEff/snpEff.jar ann -noStats ${params.ref_genome_id}
\
     ${consensus_vcf_gz} > detected_variants_consensus_annotated.vcf
```

The analysis was conducted using the snpEFF program. The only required input, besides specifying the genome name present in the database, is a VCF file containing mutations. This file is then parsed into a text file using bcftools. At this stage, a consensus VCF file is created. Note that this file is only used for functional analysis, not for creating the genome sequence.

```
bcftools query --format '%REF%POS%ALT| %ANN \n' \
           detected_variants_consensus_annotated.vcf.gz | \
              cut -d "|" -f1,3,5,12 | \
              tr "|" "\t" | \
              awk  'BEGIN {OFS = "\t"} {if ( \$2 ==
"upstream_gene_variant" || \$2 == "downstream_gene_variant")
{gene="."; aa="."} else {gene=\$3; aa=\$4}; print gene, \$1, aa, \$2}'
> detected_variants_consensus_annotated.txt
```

The `-n+2` option ensures that only mutations occurring in at least two partial files are reported, while `-c` all defines how positions in different files are treated, with "all" indicating that entries in different files covering the same position are treated as identical.

Subsequently, the snpEFF program utilizes the files `0000.vcf` and `0001.vcf` created in the dir directory, which contain VCF files from the lofreq and freebayes programs filtered to include only positions present in at least 2 programs. These files are parsed in the script to obtain a nicely formatted table.

Note: In the case of complicated variants (deletions of different lengths), there is no guarantee that such a position will be found in the annotated VCF file, which is a limitation of the isec function.

SVs calling: Module picard

```
java -jar /opt/picard/picard.jar PositionBasedDownsampleSam \
                                 --INPUT ${bam} \
                                 --OUTPUT downsample.bam -F \${NORM}
```

At this stage, we prepare a .bam file that will be used by the Manta program for SV identification. The source .bam file is the one generated in module bwa (Main: Module bwa), not module merging (Main: Module merging), because primer masking may result in the identification of additional/longer SVs than actually exist.

Picard was not designed for analyzing data from amplicon-based sequencing. Empirically, consistent results with EQA were obtained when the .bam file contained approximately 200,000 reads. To obtain a .bam file with this number of reads while preserving their genome distribution as in the original file, the `PositionBasedDownsampleSam` function was used. Apart from specifying the input and output file names, the function requires an argument (-F), i.e., a fraction (not a number) of the original number of reads to be retained in the newly created file. In the script, this fraction is symbolically denoted as NORMALIZED($max_number_for_SV) to illustrate that it depends on the value provided by the user for the max_number_for_SV variable, which defaults to 200,000.

It's worth explaining why the `PositionBasedDownsampleSam` function was not used in Step 4. This is because amplicon-based sequencing is characterized by very large coverage gaps between different, often adjacent genome regions. Thus, there are regions with coverage as high as 20,000 next to regions with significantly lower coverage, e.g., 50. In such a situation, it is impossible to reduce the coverage in a region with very high coverage to values like 1,000-5,000 while simultaneously maintaining coverage of 50 for a region with low coverage. After applying the `PositionBasedDownsampleSam` function, the coverage in low coverage regions will be much lower, often below 20, which is an important threshold because it defines regions that are masked as "N". Moreover, it may lead to erroneous SV identification in such regions. Additionally, `PositionBasedDownsampleSam` does not understand that reads come from sequencing a specific amplicon, and some reads are "more important" and carry correct information, while some reads can be filtered out because they are irrelevant.

SVs calling: Manta

The purpose of the Manta module is to identify structural variants. Manta utilizes a separate container named `nf_illumina_sars-3.0-manta`. Manta is a standalone pipeline described in detail on the GitHub page (https://github.com/Illumina/manta).

Lineages: Modules pangolin and nextclade

```
pangolin --outfile pangolin_lineage.csv \
         --threads ${params.threads} \
         ${consensus_masked_sv_fa}

nextclade run --input-dataset
/home/external_databases/nextclade_db/sars-cov-2.zip \
          --output-csv nextstrain_lineage.csv \
          --output-all nextclade_lineages \
          ${consensus_masked_sv_fa}
```

After obtaining the genome sequence of the sampled individual, each of the variants (individual sequences from each program as well as the consensus) is analyzed for classification into specific variants. For this purpose, we use the pango and nextclade programs. Their invocation does not require special arguments, as their configuration is done at the container creation level.

Model building: Module modeller

```
modpy.sh modeller_create_alignment.py ${target_fasta}
modpy.sh modeller_build_model.py alignment.pir
```

Modeller is a program used for building protein models using homology modeling method (i.e., based on the known structure of a related protein). In our case, we chose protein 7dwz (https://www.rcsb.org/structure/7DWZ) as the basis for modeling. Modeller is software that requires a license key, which needs to be provided during the container build process. As a result, it returns a PDB file containing the trimer of the S protein, as well as a file with sequence alignment. Masked regions are modeled as amino acids labeled with the symbol `UNK` without side chains (effectively as glycines).

Dokumentację formatu PDB można znaleźć pod adresem:
https://www.wwpdb.org/documentation/file-format

Statistics: Modules wgsMetrics and simpleStats

Module wgsMetrics

```
java -jar /opt/picard/picard.jar CollectWgsMetrics --
REFERENCE_SEQUENCE ${reference_fasta} \
                                                --MINIMUM_BASE_QUALITY
${params.quality_initial} \
                                                --
MINIMUM_MAPPING_QUALITY 30 \
                                                --INPUT ${bam} \
                                                --OUTPUT
picard_statistics.txt
```

Module simpleStats

```
calculate_N.py ${consensus_masked_fa} consensus_masked_N_summary.txt

 # wybrane pozycje z picard-a
 OUT=`cat ${picard_statistics_txt} | head -8 | tail -1`
 HEADER=`cat ${picard_statistics_txt} | head -7 | tail -1`

 echo -e "\${HEADER}\t\${OUT}" > picard_summary.txt

 # użycie primerów
 touch log.txt
 MES1=`primer_usage_sum.py ${params.primers} log.txt 40 | head -1`
 MES2=`primer_usage_sum.py ${params.primers} log.txt 40 | head -2 |
tail -1`

 echo -e "\${MES1}" > primers_poor_stretch.txt
 echo -e "\${MES2}" > primers_poor.txt
```

Here we parse partial results/logs from programs:

1. Counting the number of N's in the genome.

2. Summary of the percentage of the genome covered at 5x, 10x, 20x, 30x.

3. Summary of which primers had low usage.

Alphabetic list of modules

bwa.nf

coinfection_analysis.nf

coinfection_ivar.nf

coinfection_varscan.nf

consensus.nf

consensusAnalysis.nf

consensusMasking.nf

dehumanization.nf

fastqc.nf

filtering.nf

freeBayes.nf

freyja.nf

functionalAnalysis.nf

indexGenome.nf

kraken2.nf

lofreq.nf

lowCov.nf

manta.nf

masking.nf

merging.nf

modeller.nf

nextclade.nf

pangolin.nf

picard.nf

simpleStats.nf

trimmomatic.nf

varscan.nf

vcf_for_fasta.nf

viterbi.nf

wgsMetrics.nf

Pipeline customization

All files required by the module for SNP/INDEL identification are located in the directory `data/generic` if the installation was conducted exactly as described in Quickstart. During the image building process, its contents are copied into the image. To use custom files, you need to appropriately modify the contents of the subdirectories "adapters", "contaminations", "genome", "modeller", "primers" and "vcf_template" before building the image.

Adapters file

The adapter sequences are located in the subdirectory `adapters`. In cases where the length of the insert (the DNA fragment to be sequenced) is shorter than the number of cycles in sequencing, it may happen that residues of adapter sequences are present at the 3' or 5' end next to the actual insert sequence. This unnecessary part of the read should be trimmed, which requires specifying the adapter sequences used during sequencing. Besides the adapters currently distributed with the trimmomatic program, you can create your own adapter sequence files provided they are in fasta format. Path to `adapters` dir is passed as a one of a parameter. The pipeline assumes the use of adapters placed in the file `TruSeq3-PE-2.fa`.

Indexed genome

The indexed genome of SARS-CoV-2 is located in the subdirectory `data/generic/genome/SarsCov2`. After building the image, files from this directory are available inside the container in the directory /SARS-CoV2/genome/SarsCov2/. To propose your own version of the genome, follow these steps:

1. Download the genome sequences in fasta format (for example, for the SARS-CoV-2 virus, the reference sequence is available on the website https://www.ncbi.nlm.nih.gov/sars-cov-2/). Save the downloaded file with the name sarscov2.fasta in any directory.

2. Index the genome using the program BWA. Navigate to the directory where you saved the sarscov2.fasta file and execute the following command. Assuming the `bwa` program is in the `$PATH`.

   ```
   bwa index sarscov2.fasta
   ```

3. Index the genome using the program faidx. Navigate to the directory where you saved the sarscov2.fasta file and execute the following command. Assuming the `samtools` program is in the `$PATH`.

```
samtools faidx sarscov2.fasta
```

4. Copy the contents of the directory with your own indexed genome to `data/generic/genome/SarsCov2`. This way, you overwrite the existing files there.

5. Build the image. The built image will only contain the new genome. The originally used version of the genome will not be available to the container.

6. Note that functional mutation effect predictions will only work if the genome sequence has the header `MN908947.3`. To change this, modify the Dockerfile in the section starting with "#SnpEFF". Add at the end of this section, after the line `RUN java -jar snpEff.jar download MN908947.3`, your own identical command but replacing the name `MN908947.3` with the header used by your new genome. The snpEFF database is regularly updated, but it is possible that our sequence is not included in this database.

7. Your own genome also requires creating your own file with the location of primers if the genome has a different header than `MN908947.3`. Changing the genome sequence may require updating the location of primer sequences in the genome.

Primers

1. For the SARS-CoV-2 virus, two protocols based on amplicons are currently used: the dominant ARTIC protocol and the long-read Midnight protocol. Over time, new versions and modifications related to emerging virus variants appear. The primers available in May 2022 are located in the "primers" subdirectory and are accessible in the container at the path `data/generic/primers`. Each of these directories contains subdirectories `V1`, `V2`, `V3`, `V4`, `V4.1` (primers used in subsequent versions of the ARTIC protocol), and `V1200` (primers used in the Midnight protocol). Additionally, primers used in the EQA test from April 2023 are added (`SARS1_partmerge_exp` for sequencing from the "SARS1" test and `SARS2_partmerge_exp` for sequencing samples in the "SARS2" test). In each of these directories, there are two files: `nCoV-2019.scheme.bed` and `pairs.tsv`. `nCoV-2019.scheme.bed` is a file containing information about the primer locations in the reference genome, and pairs.tsv contains information about which primer pairs flank each of the resulting amplicons. **When invoking the pipeline, the path to the selected bed file with primers must be provided. There is no "default" version.**

2. When creating custom .bed files, remember that (I) the primer positions in the bed file are indexed from 0, not from 1. (II) The ranges are half-open, meaning the start position is treated as the first position in the genome that contains the primer, and the END position is treated as the first position in the genome that the primer does not cover. Below is an example of how a properly formatted bed file should look:

```
MN908947.3 29 54 nCoV-2019_1_LEFT 1 +
MN908947.3 385 411 nCoV-2019_1_RIGHT 1 -
MN908947.3 319 342 nCoV-2019_2_LEFT 2 +
MN908947.3 322 333 nCoV-2019_2_LEFT_alt 2 +
```

3. The .bed file must contain the following columns separated by tabs:

    1. Reference genome name. In case of using a genome other than MN908947.3 (point B.2), always create a primer scheme from scratch.
    2. Start position for the primer
    3. End position for the primer
    4. Primer name. The name is built according to the scheme in which consecutive elements are separated by " *": nCoV-2019* (amplikon number) *(from which side of the amplikon the primer is located, possible expressions are LEFT or RIGHT)* (optional field where we provide the expression 'alt' if a given amplikon has more than one primer flanking it from the same side). For example, `nCoV-2019_2_LEFT_alt` means it is the second primer flanking from the 5' side of amplikon number 2.
    5. Pool identifier from which the amplikon originates. It can appear as a single digit, e.g., "1" or "2", or as a unique text, e.g., `nCoV-2019_1` or `nCoV-2019_2`.
    6. The direction of the strand to which the primer hybridizes.

    Publicly available files often have column 7 with the primer sequence, but it is not required. "Basic" and "alt" primers are NOT combined unless we understand the implications for further analysis. Primers can be combined if they are duplicates, meaning they have identical values in columns 2 and 3.

4. The pairs.tsv file should contain only two columns separated by tabs: i. Name of the primer flanking the amplicon from the 5' side. The name is identical to column 4 of the .bed file. ii. Name of the primer flanking the amplicon from the 3' side. The name is identical to column

4 of the .bed file. If there are more than one primer flanking the amplicon from the same side, provide the primer that generates the longer amplicon.

5. After creating both files, place them in a directory with a unique name inside `data/generic/primers`.

6. Primers used in the Midnight protocol (referred to as `V1200`) are available as an archive on the website https://zenodo.org/record/3897530#.Xv5EFpMzadY. These files were prepared based on the protocol described at https://www.protocols.io/view/sars-cov2-genome-sequencing-protocol-1200bp-amplic-rm7vz8q64vx1/v6?step=20. Download these files by executing the following commands:

```
cd ${HOME}/my_primers/nCoV-2019
wget
https://zenodo.org/record/3897530/files/1200bp_amplicon_bed.tar.gz
tar -zxf 1200bp_amplicon_bed.tar.gz
```

7. Primers used in the ARTIC scheme are available in the repository https://github.com/artic-network/fieldbioinformatics.git

8. Due to the operation of the primer masking program, it is recommended to extend the amplicon ranges by 1bp in the 3' and 5' directions. Practically, this means that in the primer file, the START field of the LEFT primer will be one less than implied by the sequence, and the END field of the RIGHT primer will be one more.