PleEpiSeq Phylogenetic Pipeline Documentation

Michał Kadlof, Michał Łaźniewski

The document provides technical documentation for the bioinformatics processing protocols (also known as pipelines) developed in the PlEpiSeq project. These protocols are designed for the automatic analysis of multiple samples to provide an input for microreact program. It covers the architecture, installation and execution procedures, the content of the Nextflow modules, and details of the validation tests.

Spis treści

[**1.** **Quick start** 3](#_Toc202278957)

[**a.** **Prerequisites** 3](#_Toc202278958)

[**b.** **Installation** 4](#_Toc202278959)

[**c.** **Running the viral pipeline** 5](#_Toc202278960)

[**d.** **Running the bacterial pipeline** 5](#_Toc202278961)

[**2.** **Hardware requirements** 6](#_Toc202278962)

[**3.** **Repository layout** 6](#_Toc202278963)

[**4.** **Bacterial SNP pipeline – step‑by‑step** 6](#_Toc202278964)

[**5.** **Viral pipeline – step‑by‑step** 7](#_Toc202278965)

[**6.** **Output files** 7](#_Toc202278966)

[**7.** **Viral pipeline modules** 8](#_Toc202278967)

[**8.** **Bacterial pipeline modules** 8](#_Toc202278968)

[**9.** **Bacterial pipeline wrapper** 13](#_Toc202278969)

[**10.** **Viral pipeline wrapper** 13](#_Toc202278970)

# **Quick start**

# **Prerequisites**

1. GNU/Linux computing server with x86\_64 architecture and at least one GPU unit (recommended Nvidia A100 with 80Gb of VRAM or newer). The pipeline was tested on servers with following operational systems Ubuntu 20.04.06 LTS, Ubuntu 24.04.2 LTS, and Debian 12.5. Pipeline was never tested on Windows operating system and will likely not work, unless executed via WSL2 (windows subsystem linux).
2. User must have sudo privileges, and must be added to the docker group.
3. Docker (<https://docs.docker.com/desktop/install/linux-install>). The pipeline was tested on docker version 24.0.7 and 27.3.1.
4. Nexflow (<https://www.nextflow.io/docs/latest/install.html>) with the nextflow binary file located in a directory that is in the PATH variable. One can copy nextflow binary to e.g. /usr/local/bin

sudo mv nextflow /usr/local/bin

The pipeline was tested on nextflow version 24.10.3.5933, 24.04.4.5917, and 24.10.4.5934.

1. [optional] Slurm - a system workload manager (https://slurm.schedmd.com/). A program used by nextflow to distribute jobs between multiple nodes in a multiple server setup. Installation of this program is beyond the scope of this documentation. Slurm is not required to run our pipeline but it allows for independent execution of multiple nextflow jobs .
2. Nextflow config (<https://www.nextflow.io/docs/latest/config.html>). The config defines which nextflow executors are available, and adjust their default parameters to work best on a given machine. Depending on config location it scope can vary from global, available to all executions of nextflow on a given host, to local, when a given config file is used exclusively for a specific nexflow execution Below is the config file used in PZH for a server with 255 CPU cores, 2 Tb of RAM and 8 GPUs. In the example below text after “#” is a comment.

# definition of profiles accessible to nextflow

profiles {

local {

# creates a profile called “local”.

process.executor = 'local' # name of the executor used by this profile

process.errorStrategy = 'retry' # all modules in the pipeline executed using this profile will are resubmitted if they fail

process.maxRetries = 2 # a module can be resubmitted for execution up to 2 times. After that many attempts module will send an error signal to nextflow

}

slurm {

# create a profile called “slurm”. It uses nextflow’s “slurm” executor. Slurm will submit jobs to a host machine called “a100-1”

process.executor = 'slurm'

process.clusterOptions = '--nodelist=a100-1'

process.errorStrategy = 'retry'

process.maxRetries = 2

}

}

executor {

# Here we modify default behaviour of the nextflow build-in executors

$local {

cpus = 256 # local executor can allocate jobs that require that many CPUs at a time

memory = '2000 GB' # local executor can allocate jobs that require that many RAM at a time

}

$slurm {

queueSize = 500 # slurm can spawn that many processes at a time.

}

}

# “Global” modifiers

docker.enabled = true # allow execution of processes inside docker containers

# **Installation**

1. Clone the Repository with the source code

git clone <https://github.com/mkadlof/plepiseq-phylogenetic-pipeline.git>

cd plepiseq-phylogenetic-pipeline

1. Build the container images

docker build -t pzh\_pipeline\_viral-phylo -f Dockerfiles/Dockerfile-viral .

docker build -t pzh\_pipeline\_bacterial-phylo -f Dockerfiles/Dockerfile-bacterial

1. Pull the Prokka helper image

docker pull staphb/prokka:latest

1. (optional) Copy & edit the template config (see 2.6 for detailed explanation)

cp nextflow.config.template nextflow.config

# **Running the viral pipeline**

1.

2.

CODE BOX

# **Running the bacterial pipeline**

1. Create a working directory where you want to store the results.
2. Copy the nf\_pipeline\_bacterial\_phylo.sh script from the repository’s root directory into your working directory.
3. (Optional) Copy a valid metadata file e.g. metadata\_salmonella.txt file and a fastas/ directory with uncompressed genome FASTA files into the working directory. Example files are available in the data/example\_data/salmonella directory of the repository
4. Use provided shell wrapper to execute the pipeline. Shell wrapper validates input and forwards all parameters to Nextflow:

bash nf\_pipeline\_bacterial\_phylo.sh \  
 --metadata metadata\_salmonella.txt \  
 --inputDir fastas/ \  
 --inputType fasta   
 --genus Salmonella \  
 --results\_prefix Salmonella\_run1 \  
 --threads 48 \  
 --projectDir PATH\_TO\_REPO

Substitute PATH\_TO\_REPO with a valid directory where repo was cloned (2.b.1).

Call the shell script with -h to see **all** available flags. It is not recommended to use nextflow scripts directly, as all defaults are only located i the shell wrappers.

# **Hardware requirements**

The pipeline does not require GPU acceleration, however, some steps like proposing core genome with roary can be computationally heavy. Below are the minimal requirement to run the pipeline, however some modules (see chapter 7 and 8) might benefit from increasing number of available CPUs.

|  |  |  |  |
| --- | --- | --- | --- |
| Pipeline | Suggested CPU | RAM | GPU |
| **Bacterial** | 12 | 60 GB | 0 |
| **Viral** | 12 | 30 GB | 0 |

# **Repository layout**

The structure of the main repository with its crucial components is outlined below.

├── Dockerfiles/   
├── modules/   
├── bin/   
├── dag\_png/   
├── data/   
├── nf\_\*\_pipeline\*.nf   
└── nf\_pipeline\_bacterial\_phylo.sh

1. Dockerfiles - a directory with two Dockerfiles (Dockerfile-bacterial , Dockerfile-viral) used to create a suitable environment to execute nextflow modules
2. Modules – a directory with all the modules used by the viral pipeline. For bacterial pipeline modules are part of the corresponding .nf file
3. bin – a directory with a number of helper scripts used by the pieplines
4. dag\_png – a directory with figures showing the pipelines topology shown as a Sirect Acyclic Graph
5. nf\_\*\_pipeline\*.nf and nf\_pipeline\_\*\_phylo.sh – main scripts with the pipeline and corresponding shell wrappers

# **Bacterial SNP pipeline – step‑by‑step**

The workflow encoded in nf\_bacterial\_phylogenetic\_pipeline.nf proceeds as follows:

1. **Gene prediction** — annotates each genome FASTA with Prokka. This step is skipped if user already provided gff files (that is part of the Plepiseq WGS pipeline)
2. **Pangenome & core alignment** – gff files are used to predict a pangenome for analyzed bactria using Roary program. Only genes present in more than 95% of samples are kept and used to propose full genome for each sample. Proposed genomes are already aligned.
3. **Sequence indexing** – part of the augur rpipeline - prepares an index for each sample with number of ambiguous and non-standard aminoacids in the genome
4. **Quality filtering**  – part of the augur pipeline removes low‑quality strains based on Ns/ambiguities thresholds.
5. **SNP alignment & partitions** –- a custom python script is used to extract only SNPs from the samples alignment and to calculate STAM correction to GTR+G model.
6. **Duplicate handling** – identification of samples with identical genome. The redundant samples are removed prior to phylogentic tree calculations (but are later reattach so the user can see all the samples he requested)
7. **RAxML‑NG**  – Maximum likelihood tree calculations with true bootstraps. User can select custom model supported by RAxML (GTR+G+ASAM is the default).
8. **Tree refinement** – midpoint rooting tree , collapsing weak branches (branches with support below 80 are collapsed), time‑scaling with TreeTime using either user-provided evolutionary rates or values predicted by Timetree based on the provided phylogentic tree and metadata
9. **Geographic enrichment** – enrichment of user provided metadata with longitude and latitude values for better visualization.
10. **Visual exports** a Microreact project file is created it includes both user provided metadata as well ac classical phylogentic tree and timetree..

# **Viral pipeline – step‑by‑step**

TBD

# **Output files**

| File | Description |
| --- | --- |
| \*\_classical\_tree.nwk | RAxML tree, branch lengths in substitutions. |
| tree3\_timetree.nwk | Time‑scaled tree (bacterial). |
| \*\_timetree.json | Auspice JSON for visualisation. |
| \*\_microreactproject.microreact | Ready‑to‑upload Microreact project. |
| branch\_lengths\*.json, traits.json | Node annotation tables used by Auspice. |
| trace.tsv, report.html | Nextflow execution report (enable with -with-trace/-with-report). |

# **Viral pipeline modules**

TBD

# **Bacterial pipeline modules**

Below is the detailed list of all modules used by the bacterial pipeline, its name, key program, role of the module, output and resources requested by this module

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module / process | Tool & key options | Function | Input | outputs | Resources (CPUs \ RAM \ time limit) |
| **run\_prokka** | prokka | Annotate each assembly to produce GFF file | FASTA file (can be gziped) | GFF file for a sample | ≤25 CPU / 10 GB / 20 m |
| **run\_roary** | roary -i 95 -cd 95 … -i - minimum percentage identity for blastp  -cd percentage of isolates a gene must be in to be core | Create core‑gene alignment across samples. Only genes present in >95% of isolates are considered as “core” genes”. The output is the FASTA with aligned genomes of provided samples and embl file with information regarding particular gene location with the alignment | A directory with any number of GFF files | core\_genes\_alignment.fasta  core\_genes\_alignment.embl | ≤25 CPU / 30 GB / 1 h |
| **augur\_index\_sequences** | A simple module executing “augur index” | Index for downstream augur tasks. The csv file simply provides an infor which residues (standard, non-standard, Ns deletions) are present for each sample in the alignment | FASTA file with aligned genomes of analyzed samples | index.csv | 1 CPU / 30 GB / 1 h |
| **augur\_filter\_sequences** | Modules executes two independent programs “identify\_low\_quality\_sequences.py” and “identify\_low\_quality\_sequences.py” | identify\_low\_quality\_sequences.py script is used to parse index csv and select based on provided threshold (for number of Ns and ambiguous residues) samples that do not pass QC criteria. This samples are next excluded from a FASTA file usind “augur filter” command. The FASTA file with aligned sequences of valid samples is called “valid\_sequences.fasta” | Index csv,  FASTA with aligned genomes, embl files with gene location | valid\_sequences.fasta | 1 CPU / 30 GB / 1 h |
| **prepare\_SNPs\_alignment** | prep\_SNPs\_alignment\_and\_partition.py | Module is a wrapper around prep\_SNPs\_alignment\_and\_partition.py. The script extract from the provided alignment only variable sites creating an alignment build solely of SNPs. To get a valid branches length it calculates ASAM correction for GTR+G model. ASAM correction is simply a number of constant sites in the alignment split across 4 standard nuclotides | Fasta file with alignes valid samples, embl file with genes location in the alignment file | alignment\_SNPs.fasta, partition.txt | As specified by user (no upper limit) / 10 GB / 2 h |
| **identify\_identical\_sequences** | find\_identical\_sequences.py | A wrapper around find\_identical\_sequences.py which identifies samples with identical sequence. Create a simple text file alignment\_SNPs\_ident\_seq.csv where each row represent unique sequence. Row is a tab separated and in case multiple samples share the same sequence samples id are listed in tab separated format. Next from input fasta files is filtered so only samples with id at the firs position are kept | Fasta file with the alignment of samples passing QC criteria  Partition file | alignment\_SNPs\_unique.fasta, mapping CSV | As specified by user (no upper limit) / 10 GB / 2 h |
| **run\_raxml** | raxml-ng with following parameters --precision 15 (branches length ar rounded to this decimal numbers)  -- all ( perform ML search + bootstrapping)  --site-repeats on (use site repeats optimization default is “on”)  --model GRT+G+ASAM  --tree 10 - number randomly generated ostarting trees  --bs-trees 200 – number of bootstrap replicates  --brlen scaled (branch length linkage between partitions scaled is the default) | Wrapper around raxml-ng main program to predict phylogenetic tree based on the provided fasta file with SNPs alignment. We dropped the idea of using per-gene partiotion file to speed up calculation several times (at the cost of reduced output quality) | SNPs alignment of samples  Partition file with model used | tree.raxml.support | As specified by user (no upper limit) / 10 GB / 8 h |
| **restore\_identical\_sequences** | root\_collapse\_and\_add\_identical\_seq\_to\_tree.py | Post processing of raxm-ng generated trees. This part includes several steps:   * 1. Tree rooting (midpoint method)   2. Collapse branches if their support is below 70   3. Reintroduce identical sequences (leaf representing identical sequence is turned into node from which all identical sequences arise, all of appended sequences have branch length 0) | Nwk file with raxm-ng proposed tree  A text file generated with **identify\_identical\_sequences** | ${input\_prefix}\_classical\_tree.nwk | 1 CPU / 10 GB / 10 m |
| **add\_temporal\_data** | augur refine. Most parameters are degfualts, however we add  --keep-polytomies (to adjust for reintroduction of identical sequences that creates polytomies) | A wrapper around augur refine that itself is a wrapper around the treetime program. This program take raxml-ng propose topology and alters branches lengths to reflect temporal signal for leafs (taken from metadata) and predicted occurrence of a node (predicted with treetime). The evolutionary rate for analyzed samples can be (I) user provided (II) either estimated from the data by treetime (III) hardcoded based on available data for a given genus. Priority is assigned to user provided value , than treetime predicted value, and if the data do not allow to predict reliable temporal signal, to value obtained from publication | Nwk file with phylogenetic tree  Metadata (for auspice)  Alignment (for auspice) | tree3\_timetree.nwk, traits.json, tree3\_rescaled.nwk | 1 CPU (treetime s not parallelized) / 20 GB / 5 h |
| **metadata\_for\_microreact** | same script as viral branch | Prepare Microreact metadata (country vs city granularity) |  | ${input\_prefix}\_metadata\_microreact.tsv | 1 CPU / 20 GB / 1 h |
| **prepare\_microreact\_json** | prepare\_json\_for\_microreact.py | Combine rescaled tree & metadata into .microreact bundle |  | ${input\_prefix}\_project.microreact | 1 CPU / 20 GB / 1 h |

# **Bacterial pipeline wrapper**

Below is the list of all oprions that can be specified with shell wrapper:

TBD

# **Viral pipeline wrapper**

Below is the list of all oprions that can be specified with shell wrapper:

TBD