Contents

NOAA/AOML Amplicon Sequence Processing Protocol using Tourmaline 2 (Eukarya-18S-V9-Lane-Medlin)

1-BACKGROUND

1 Methods

Tourmaline builds on the latest methods for analysis of microbial and eDNA amplicon sequence data. This section describes those methods and provides tutorials for some of them.

1.1 Amplicon sequence variants

Amplicon sequencing (metabarcoding) is a method whereby a single DNA locus in a community of organisms is PCR-amplified and sequenced. Two methods of amplicon sequence processing are supported, both of which generate ASVs (amplicon sequence variants), which approximate the “true” or “exact” sequences in a sample, rather than OTUs (operational taxonomic units), which blur sequencing errors and microdiversity through clustering:

- DADA2 implements a quality-aware model of Illumina amplicon errors to infer sample composition by dividing amplicon reads into partitions consistent with the error model (Callahan et al., 2016).
- Deblur is a greedy deconvolution algorithm based on known Illumina read error profiles (Amir et al., 2017).

1.2 QIIME 2

QIIME 2 (Bolyen et al., 2019) is one of the most popular amplicon sequence analysis software tools available. It supports both DADA2 and Deblur denoising algorithms as well as a variety of downstream diversity and statistical analyses and visualizations. Click here for a tutorial on QIIME 2.

1.3 Snakemake

Snakemake is a workflow management software that allows for reproducible and scalable workflows for bioinformatics and other data analyses. It keeps track of input and output files, storing output files in a logical directory structure. It uses rules to define commands and only runs rules when they are required to produce the desired output. Click here for a tutorial on Snakemake.

2 Assessing your data

Before starting any bioinformatics workflow, it is important to assess your data and metadata to decide how they need to be formatted and inform your parameter choices. The questions below can help determine the best parameters for processing and to be able to evaluate the success of your completed workflow.

2.1 Amplicon locus

- What is the locus being amplified, and what are the primer sequences?
- How much sequence variation is expected for this locus (and primer sites) and dataset?
- Is the expected sequence variation enough to answer my question?
- What is the expected amplicon size for this locus and dataset?

2.2 Sequence data

- What type and length of sequencing was used? (e.g., MiSeq 2x150bp)
- Were all my samples sequenced in the same sequencing run? (Rule `check_illumina_run` will check for this.)
- Do I have long enough sequening to do paired-end analysis, or do I have to do single-end analysis only?
- Has the relevant sequence pre-processing been done already: Demultiplexing? Quality filtering and/or trimming? Primer removal? FastQC/MultiQC profiling before and/or after filtering/trimming? (Note on quality trimming: If you plan on using DADA2 for denoising, the developers recommend no quality trimming be done before running, as it confuses the error profiling algorithm of DADA2. Quality filtering including complete removal of erroneous sequences is still advisable.)

2.3 Sample set and metadata

- Is my metadata file complete? Are the relevant parameters of my dataset present as numeric or categorical variables?
- Do I have enough samples in each group of key metadata categories to determine an effect?

3 Workflow overview

The table below describes the basic steps of the Tourmaline workflow. Further instructions are provided in the sections Setup and Run.

In the file paths below, `{method}` is one of:

- `dada2-pe` (paired-end DADA2)

- `dada2-se` (single-end DADA2)
- `deblur-se` (single-end Deblur)

and `{filter}` is one of:

- `unfiltered` (representative sequences and feature table *are not* filtered by taxonomy or feature ID)
- `filtered` (representative sequences and feature table *are* filtered by taxonomy or feature ID)

Step	Command	Output
Format input and configuration file	(ad hoc)	`config.yaml`, `Snakefile`, `00-data/metadata.tsv`, `00-data/manifest_se.tsv`, `00-data/manifest_pe.tsv`, `00-data/refseqs.fasta` or `01-imported/refseqs.qza`, `00-data/reftax.tsv` or `01-imported/reftax.qza`
Import data	`snakemake {method}_denoise`	`01-imported/` (multiple files)
Denoising	`snakemake {method}_denoise`	`02-output-{method}/00-table-repseqs/` (multiple files)
Taxonomic assignment	`snakemake {method}_taxonomy_{filter}`	`02-output-{method}/01-taxonomy` (multiple files)
Representative sequence curation	`snakemake {method}_diversity_{filter}`	`02-output-{method}/02-alignment-tree` (multiple files)
Core diversity analyses	`snakemake {method}_diversity_{filter}`	`02-output-{method}-{filter}/03-alpha-diversity/` `02-output-{method}-{filter}/04-beta-diversity/` (multiple files)
Report	`snakemake {method}_report_{filter}`	`03-reports/report_{method}_{filter}.html`

2-SETUP REQUIREMENTS

1 Installing dependencies

Tourmaline 2 requires the following software:

- Conda or Mamba
- QIIME 2 version 2024.10 (amplicon workflow)
- Snakemake
- Python packages: biopython, biopython, yq, parallel
- Multiple sequence alignment tools: Muscle v.3.8 and Bowtie 2

2 Installation options

2.1 Option 1: Native installation

The native installation builds on the Conda installation of QIIME 2.

2.1.1 Conda or Mamba First, if you don't have Conda or mamba installed on your machine, install Miniconda for your operating system (Python 3.8+ version). Once Miniconda is installed, you can install Mamba (a faster drop-in replacement for Conda) with the following command:

```
conda install -n base -c conda-forge mamba
```

All the following commands can then be run using either conda or mamba.

2.1.2 Snakemake Second, install Snakemake into its own environment.

```
conda create -c conda-forge -c bioconda -n snakemake-tour2 snakemake biopython yq parallel
```

Your conda environment snakemake-tour2 must be activated:

```
conda activate snakemake-tour2
```

2.1.3 QIIME 2 Third, install QIIME 2 (2024.10) amplicon workflow in a Conda environment, if you haven't already. See the instructions at qiime2.org.

```
conda env create -n qiime2-amplicon-2024.10 --file https://data.qiime2.org/distro/amplicon/qiime2-amplic
```

2.1.4 Multiple sequence alignment tools Third, install the multiple sequence alignment tools Muscle v.3.8 and Bowtie 2 in a conda environment, if you haven't already.

```
conda create -c conda-forge -c bioconda -n bt2-blca biopython muscle=3.8 bowtie2
```

2.1.5 Tourmaline 2 The first time you run Tourmaline 2, you will clone (download) a copy of the development branch of the Tourmaline GitHub repository (including its directory and contents) to your computer.

Navigate to the directory where you want to clone and store the Tourmaline repository. If you are using the Docker container, a good location for your project directory is /data.

```
WORKINGDIR = /absolute/path/to/user/directory
cd $WORKINGDIR # or your working directory
git clone --branch develop https://github.com/aomlomics/tourmaline.git
```

2.2 Option 2: Docker container

An alternative to the native installation is a Docker container. Here are the steps to follow:

To load the image on a computer with Docker installed:

```
docker load < tourmaline2-dev-amd64.tar.gz
```

To see the image:

```
docker images
```

Run the local image in a container:

```
docker run -v $HOME:/data -it tourmaline2-dev-amd64
```

3-SET UP TOURMALINE 2 INPUTS

This section describes one way to organize your data and use the `00_setup_tourmaline2_analyses.py` script in `tourmaline/scripts` to prepare your data and configuration files for a Tourmaline2 analysis.

Alternatively, users can create their own manifest and metadata files, which allows for more flexibility—for example, sample names that do not necessarily match the names of the raw FASTQ files.

1 Raw data

- Organize your raw paired-end FASTQ files in subfolders under a main directory.
- Files should be named with `_R1` and `_R2` (e.g., `Sample1_R1.fastq.gz`, `Sample1_R2.fastq.gz`).

Directory structure example:

```
WORKINGDIR = /absolute/path/to/user/directory

$WORKINGDIR/raw_reads/
    seq_run_name_1/
         Sample1_R1.fastq.gz
         Sample1_R2.fastq.gz
         Sample2_R1.fastq.gz
         Sample2_R2.fastq.gz
         Sample3_R1.fastq.gz
         Sample3_R2.fastq.gz
    seq_run_name_2/
          Sample1_R1.fastq.gz
          Sample1_R2.fastq.gz
          Sample2_R1.fastq.gz
          Sample2_R2.fastq.gz
```

`seq_run_name`: see definition below.

2 Script overview

The script automates the following steps:

1. Organizes raw FASTQ files into output folders for each subfolder in the raw read folder.
2. Generates manifest files (with options to force regeneration or use existing files).
3. Copies and renames configuration files for each Tourmaline step.
4. Updates config files with correct paths and parameters for each run.
5. Prepares your data for reproducible, automated amplicon data processing with Tourmaline 2.

3 Command-line usage and options

```
python $WORKINGDIR/tourmaline/scripts/00_setup_tourmaline2_analyses.py \
  --read-folder <PATH_TO_RAW_READS> \
  --working-directory <PATH_TO_OUTPUT> \
  --data-type <DATA_TYPE> \
  --tourmaline2-folder <PATH_TO_TOURMALINE2_CONFIGS> \
  --run-name <RUN_NAME> \
  --primers <FWD_PRIMER/REV_PRIMER> \
  --classifier-method <naive-bayes|consensus-blast|consensus-vsearch|bt2-blca> \
  --database <PATH_TO_DB> \
  [--metadata-folder <PATH_TO_METADATA>] \
  [--config-file <CONFIG_FILE>] \
  [--force] \
  [--threads <N>]
```

- `--read-folder <PATH_TO_RAW_READS>` (**required**): Path to the folder containing seq_run_name folders of raw reads. Each seq_run_name folder is treated as a batch/analysis.
- `--working-directory <PATH_TO_OUTPUT>` (**required**): Path where output folders and files will be created.
- `--data-type <DATA_TYPE>` (**required**): Data type (e.g., `COI`, `12S`, `18S`, `28S`, etc.). This should the be a subfolder of the seq_run_name folder.
- `--tourmaline2-folder <PATH_TO_TOURMALINE2_GITHUB_REPOSITORY>` (**required**): Path to the folder containing original Tourmaline2 config files.

- `--run-name <RUN_NAME>` (**optional**, default: empty string): String to append to output files for identification. If not provided, only the seq_run_name folder name and data type are used.
- `--primers <FWD_PRIMER/REV_PRIMER>` (**optional**, default: empty string): Forward and reverse primer sequences, separated by `/`.
- `--classifier-method <METHOD>` (**required**): Taxonomy classification method. Choose one of:
  - `naive-bayes`
  - `consensus-blast`
  - `consensus-vsearch`
  - `bt2-blca`
- `--database <PATH_TO_DB>` (**required**): Path to the reference database or classifier file. The script will extract the base name to rename `00_config_03_taxonomy.yaml`. *Notes: Provide a descriptive name for the reference database used, no spaces (e.g., original file name, source and version). Ex: database_name: silva-138-99-515-806_q2-2021.2*
- `--config-file <CONFIG_FILE>` (**optional**, default: `all`)
  - Specify which config file to update/copy:
    - `00_config_01_qaqc.yaml`
    - `00_config_02_repseqs.yaml`
    - `00_config_03_taxonomy.yaml`
    - `all` (default: copies and updates all three)
- `--force` (**optional**): Force regeneration of manifest and metadata files even if they already exist. Otherwise, existing files are reused.
- `--threads <N>` (**optional**, default: 6): Number of threads to use for config updates and processing.

---

4 Command-line example

```
WORKINGDIR = /absolute/path/to/user/directory

python $WORKINGDIR/tourmaline/scripts/00_setup_tourmaline2_analyses.py \
  --read-folder $WORKINGDIR/raw_reads \
  --working-directory $WORKINGDIR/18Sv9_analyses \
  --data-type 18S \
  --tourmaline2-folder $WORKINGDIR/tourmaline \
  --run-name 18Sv9_20250728 \
  --primers GTACACACCGCCCGTC/TGATCCTTCTGCAGGTTCACCTAC \
  --classifier-method naive-bayes \
  --database $WORKINGDIR/Databases/silva-138-99-515-806_q2-2021.2-classifier.qza \
  --threads 36
```

---

5 Output folders

For each seq_run_name folder in your `raw_reads` directory, a corresponding folder is created at `working_directory/seq_run_name`.

6 Manifest file

A manifest file (`00_manifest_pe.csv`) is created in the output folder, listing all FASTQ files, their sample IDs, and read directions. If the file exists and `--force` is not set, it is reused.

sample-id	absolute-filepath	direction
Sample1	$WORKINGDIR/raw_reads/seq_run_name_1/Sample1_R1.fastq.gz	forward
Sample1	$WORKINGDIR/raw_reads/seq_run_name_1/Sample1_R2.fastq.gz	reverse
Sample2	$WORKINGDIR/raw_reads/seq_run_name_1/Sample2_R1.fastq.gz	forward
Sample2	$WORKINGDIR/raw_reads/seq_run_name_1/Sample2_R2.fastq.gz	reverse
Sample3	$WORKINGDIR/raw_reads/seq_run_name_1/Sample3_R1.fastq.gz	forward
Sample3	$WORKINGDIR/raw_reads/seq_run_name_1/Sample3_R2.fastq.gz	reverse

Notes: Tourmaline also accepts other manifest formats. If you need a different format (e.g., for single-end data or the QIIME2 tab-separated format), you must create the manifest file yourself and update the corresponding config files to point to it. ***If you require a manifest format other than the default, you are responsible for creating it and updating the relevant config files to reference your custom manifest.***

Supported manifest formats (from Tourmaline documentation):

- **Tab-separated (QIIME2 standard):**
  - *Paired-end:*
    ```
    sample-id  forward-absolute-filepath     reverse-absolute-filepath
    sample1    /path/to/sample1_R1.fastq.gz  /path/to/sample1_R2.fastq.gz
    ```
  - *Single-end:*
    ```
    sample-id  absolute-filepath
    sample1    /path/to/sample1_R1.fastq.gz
    ```
- **CSV (legacy, as generated by this script):**
  - *Paired-end:*
    ```
    sample-id,absolute-filepath,direction
    sample1,/path/to/sample1_R1.fastq.gz,forward
    sample1,/path/to/sample1_R2.fastq.gz,reverse
    ```
  - *Single-end:*
    ```
    sample-id,absolute-filepath
    sample1,/path/to/sample1_R1.fastq.gz
    ```
- **No manifest file:**
  - If your FASTQ files are named using the conventions `{sample}_R1.fastq.gz` and `{sample}_R2.fastq.gz` (or `{sample}_R1_001.fastq.gz`), Tourmaline can sometimes infer sample names without a manifest.

7 Config files

Config files `00_config_01_qaqc.yaml`, `00_config_02_repseqs.yaml`, `00_config_03_taxonomy.yaml`are copied from the Tourmaline 2 GitHub repository at `working_directory/seq_run_name` and renamed with run-specific identifiers. Naming patterns: - `00_config_01_qaqc_<RUN_NAME>.yaml` - `00_config_02_repseqs_<RUN_NAME>.yaml` - `00_config_03_taxonomy_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAM`

If `--config-file` is set, only that config is copied/updated.

The script updates key fields in each config file to reflect the current run, output locations, manifest/metadata files, and processing parameters. Below are the main fields updated for each config:

7.1 00_config_01_qaqc.yaml

The script updates the following fields in the `00_config_01_qaqc_<RUN_NAME>.yaml` file:

```
sample_manifest_file: <absolute_path_to_manifest>
run_name: 01_<RUN_NAME>
output_dir: <output_dir>
```

```
trimming_threads: <threads>
# If primers are provided:
fwd_primer: <forward_primer>
rev_primer: <reverse_primer>
```

Explanation: - `sample_manifest_file`: The absolute path to the manifest file for this run. - `run_name`: Set to `01_<RUN_NAME>` for this step. - `output_dir`: The absolute path to the output directory for this run. - `trimming_threads`: Number of threads to use for trimming, as specified by the `--threads` argument. - `fwd_primer`/`rev_primer`: If primers are provided, these are set to the forward and reverse primer sequences, respectively.

7.2 00__config__02__repseqs.yaml

The script updates the following fields in the `00_config_02_repseqs_<RUN_NAME>.yaml` file:

```
run_name: 02_<RUN_NAME>
output_dir: <output_dir>
sample_metadata_file: <absolute_path_to_metadata>
sample_run_name: null
asv_threads: <threads>
fastq_qza_file: <output_dir>/01_<RUN_NAME>-qaqc/01_<RUN_NAME>_fastq.qza
```

Explanation: - `run_name`: Set to `02_<RUN_NAME>` for this step. - `output_dir`: The absolute path to the output directory for this run. - `sample_metadata_file`: The absolute path to the metadata file for this run. - `sample_run_name`: Set to `null` (not used in this workflow). - `asv_threads`: Number of threads to use for ASV processing, as specified by the `--threads` argument. - `fastq_qza_file`: Path to the QIIME2 artifact generated in the QA/QC step, constructed using the output directory and run name. - ### 00_config_03_taxonomy.yaml The script updates the following fields in the `00_config_03_taxonomy_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME>.yaml` file:

```
run_name: 03_<output_subfolder>_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME>
output_dir: <output_dir>
sample_metadata_file: <absolute_path_to_metadata>
classify_threads: <threads>
repseqs_run_name: null
repseqs_qza_file: <output_dir>/02_<RUN_NAME>-repseqs/02_<RUN_NAME>-repseqs.qza
table_qza_file: <output_dir>/02_<RUN_NAME>-repseqs/02_<RUN_NAME>-table.qza
classify_method: <classifier_method>
database_name: <database_name>
# Depending on classifier_method:
# For consensus-blast/vsearch:
refseqs_file: <absolute_path_to_db>-seqs.qza
taxa_file: <absolute_path_to_db>-tax.qza
# For naive-bayes:
pretrained_classifier: <absolute_path_to_db>
refseqs_file: null
taxa_file: null
# For bt2-blca:
bowtie_database: <absolute_path_to_db>
```

Explanation: - `run_name`: Set to `03_<output_subfolder>_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME>` for this step, encoding the seq_run_name, classifier, database, and run name. - `output_dir`: The absolute path to the output directory for this run. - `sample_metadata_file`: The absolute path to the metadata file for this run. - `classify_threads`: Number of threads to use for classification, as specified by the `--threads` argument. - `repseqs_run_name`: Set to `null` (not used in this workflow). - `repseqs_qza_file`: Path to the representative sequences artifact from the previous step. - `table_qza_file`: Path to the feature table artifact from the previous step. - `classify_method`: The taxonomy classification method used for this run. -

`database_name`: The base name of the database file (without `.qza` extension). - `refseqs_file/taxa_file`: For `consensus-blast` or `consensus-vsearch`, these are set to the database path with `-seqs.qza` and `-tax.qza` suffixes, respectively. - `pretrained_classifier`: For `naive-bayes`, this is set to the absolute path to the classifier file, and `refseqs_file/taxa_file` are set to `null`. - `bowtie_database`: For `bt2-blca`, this is set to the absolute path to the Bowtie2 database.

7.3 Directory structure example

```
WORKINGDIR = /absolute/path/to/user/directory

WORKINGDIR/
     seq_run_name_1/
          00_manifest_pe.csv
          00_metadata.tsv
          00_config_01_qaqc_<RUN_NAME>.yaml
          00_config_02_repseqs_<RUN_NAME>.yaml
          00_config_03_taxonomy_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME>.yaml
```

4-RUN TOURMALINE 2

1 STEP 1: QA/QC

Now that all input files and configuration files are set up, you are ready to run the Tourmaline pipeline for the QA/QC step.

1.1 Step 1 (QA/QC) overview

The QA/QC step is responsible for preparing your raw sequencing data for downstream analysis. The main actions performed in this step are:

- **Trimming (optional):** If specified in the config, primers and low-quality bases are trimmed from the reads using Cutadapt 5.1.
- **Merging (for paired-end):** If enabled, paired-end reads are merged after trimming, using QIIME 2 vsearch merge-pairs.
- **Quality summarization:** The pipeline generates QIIME2 visualizations (`.qzv`) summarizing the quality of raw, trimmed, and merged reads.
- **Manifest creation:** If needed, new manifest files are generated for downstream steps.

1.2 Running Step 1 QA/QC

There are two main ways to execute Tourmaline.

1.2.1 Command-Line Execution You can run Tourmaline directly from the command line using the `tourmaline.sh` script:

```
./tourmaline.sh --step qaqc \
--configfile working_directory/subfolder<NAME>/00_config_01_qaqc_<RUN_NAME>.yaml \
--cores <threads>
```

Or using short flags:

```
./tourmaline.sh -s qaqc \
-c working_directory/subfolder<NAME>/00_config_01_qaqc_<RUN_NAME>.yaml \
-n <threads>
```

- `--step/-s`: The pipeline step to run (here, `qaqc`).
- `--configfile/-c`: Path to the config file for this run.

- `--cores/-n`: Number of threads/cores to use for this run.

Note: The number of threads/cores should match the value set in your config file (e.g., `trimming_threads`). If they differ, the command-line value may use the value provided in the config file.

1.2.2 SLURM Array for Parallel Processing

The `sbatch_tourmaline2_step1.sh` script located at `tourmaline/scripts/` is designed to run the Tourmaline QA/QC step in parallel across multiple samples, batches or seq_run_name using a SLURM job array. **Before running this script, you must configure the SLURM script to match your cluster environment and project requirements.**

1.2.2.1 `sbatch_tourmaline2_step1.sh` script configuration

At the top of the script, below, you will find lines beginning with `#SBATCH`. These control how your job is submitted to the SLURM scheduler. Key directives include:

- `--mail-user` and `--mail-type`: Set your email address and notification preferences.
- `-A`, `-p`, `-q`: Set your allocation/account, partition, and queue.
- `-N`, `-n`: Number of nodes and tasks (cores) per job.
- `-t`: Maximum wall time for each job.
- `-J`: Job name.
- `--array`: Controls the number of parallel jobs (e.g., `0-7%8` runs 8 jobs at a time).
- `-o`: Output file for SLURM logs (can be customized or set to `/dev/null`).

Edit these lines as needed to match your institution's SLURM setup, your allocation, and the scale of your analysis.

In addition to the SLURM configuration, you may need to edit other variables in the script to match your environment: - The path to the Tourmaline GitHub folder below `# Change to Tourmaline GitHub directory` - The path to your Conda installation and the name of the Tourmaline environment (e.g., `conda activate snakemake-tour2`) below `# Load Conda environment manager` and `# Activate Tourmaline environment`, respectively - Any other environment-specific variables or module loads required by your cluster

Review and update these paths and commands as needed before submitting the script.

1.2.2.2 Submit SLURM script for parallel jobs

After configuring, you can submit the script as follows:

```
sbatch sbatch_tourmaline2_step1.sh \
--working-directory <working_directory> \
--config-file 00_config_01_qaqc_<RUN_NAME>.yaml
```

- `--working-directory`/`--wd`: The directory containing all seq_run_name folders and files created by `00_setup_tourmaline2_analyses.py` script.
- `--config-file`: The config file name to use for each sample.

The SLURM script will: - Launch one job per seq_run_name in your working directory (using the SLURM array mechanism). - Automatically set up output logging and skip seq_run_name that already have output. - Activate the correct Conda environment and run Tourmaline for each seq_run_name in parallel.

1.2.2.3 Monitoring SLURM Jobs and Outputs

- You can monitor your SLURM jobs with:

  `squeue -u <your_user_id>`

  Replace `<your_user_id>` with your actual username. This will show all your jobs in the queue, their status, and job IDs.

- Other useful SLURM commands:

  - `scontrol show job <job_id>`: Show detailed info for a specific job.
  - `scancel <job_id>`: Cancel a running job.
  - `sacct -j <job_id>`: Show accounting info for a completed job.
- SLURM output logs:
  - Each job will write its output to a log file, typically named like `<RUN_NAME>_tourmaline2_qaqc_<SLURM_JOB_ID>_s` These logs contain all standard output and error messages from the job, including Tourmaline progress and any errors.

1.3 Step 1 QAQC Outputs

- Inside each seq_run_name folder (in the working directory), step 1 qaqc creates an output folder (e.g., `01_seq_run_name_1_<RUN_NAME>-qaqc/`).
- Inside this output folder, you will find the results of the step 1 qaqc, including processed QIIME2 artifacts and summary files.
- If the output folder already exists, the script will skip reprocessing that sample.
- **Output structure:** All outputs are organized in a seq_run_name specific output folder, including:
- Imported QIIME2 artifacts (`.qza`)
- Quality summary visualizations (`.qzv`)
- Trimming/merging statistics and logs
- Archived config file for reproducibility

```
WORKINGDIR = /absolute/path/to/user/directory

WORKINGDIR/
    seq_run_name_1/
        seq_run_name_1_Output/
            01_seq_run_name_1_<RUN_NAME>-qaqc/
                01_seq_run_name_1_<RUN_NAME>_fastq.qza
                01_seq_run_name_1_<RUN_NAME>-qaqc_config.yaml
                raw_fastq.qza
                stats/
                    cutadapt_summary.txt
                    fastq_summary.qzv
                    raw_fastq_summary.qzv
```

2 STEP 2: REPSEQS

After completing the QA/QC step, you are ready to run the Tourmaline pipeline for the RepSeqs step, which performs denoising and generates representative sequences.

2.1 Step 2 (REPSEQS) overview

The repseqs step is responsible for denoising your sequencing data and generating representative sequences for downstream analysis. The main actions performed in this step are:

- **Denoising:** The pipeline assesses error modeling and sample inference to identify true biological sequences, using DADA2 (paired-end or single-end).
- **Filtering (optional):** If enabled, sequences can be filtered by length, abundance, prevalence, and frequency thresholds
- **Feature table generation:** Creates a feature table with ASV/OTU counts per sample
- **Quality assessment:** Generates comprehensive statistics and visualizations including:
  - Denoising statistics (DADA2 or Deblur)
  - Feature table summaries (samples and features)
  - Representative sequence visualizations
  - Sequence length distributions

- **Diversity analysis:** alpha and beta diversity metrics with rarefaction curves
- **Output formats:** Produces QIIME2 artifacts (`.qza`), visualizations (`.qzv`), and exportable formats (`.tsv`, `.biom`, `.fasta`)

Notes: The pipeline supports multiple denoising methods: DADA2 (paired-end or single-end) or Deblur.

The pipeline automatically handles the input from the previous QA/QC step and applies the appropriate denoising method based on your configuration.

2.2 `00_config_02_repseqs_<RUN_NAME>.yaml` Configuration

The `00_setup_tourmaline2_analyses.py` script automatically populates the `00_config_02_repseqs_<RUN_NAME>.yaml` file with common settings for all seq_run_name analyses. However, you should review all fields based on your specific data characteristics and research objectives before running the pipeline.

2.2.1 Key Configuration Fields Basic Settings: - `run_name`: Automatically set to `02_<RUN_NAME>` by the setup script - `output_dir`: Automatically set to the absolute path of your output directory - `sample_metadata_file`: Path to your metadata file (automatically set) - `fastq_qza_file`: Path to the QIIME2 artifact from Step 1 (automatically set) - `asv_threads`: Number of threads for ASV processing (set via `--threads` argument)

ASV Method Selection: - `asv_method` - Default: dada2pe - Options: dada2pe | dada2se | deblur - source_file: 00_config_02_repseqs_.yaml - source_term: asv_method

DADA2 Parameters (for paired-end data): - `dada2_trunc_len_f`: Forward read truncation length - Default: 245 - source_file: 00_config_02_repseqs_.yaml - source_term: dada2_trunc_len_f - `dada2pe_trunc_len_r`: Reverse read truncation length - Default: 190 - source_file: 00_config_02_repseqs_.yaml - source_term: dada2pe_trunc_len_r - `dada2_trim_left_f`: Forward read trim from left - Default: 0 - source_file: 00_config_02_repseqs_.yaml - source_term: dada2_trim_left_f - `dada2pe_trim_left_r`: Reverse read trim from left - Default: 0 - source_file: 00_config_02_repseqs_.yaml - source_term: dada2pe_trim_left_r - `dada2_max_ee_f`: Forward read maximum expected errors - Default: 2 - source_file: 00_config_02_repseqs_.yaml - source_term: dada2_max_ee_f - `dada2pe_max_ee_r`: Reverse read maximum expected errors - Default: 2 - source_file: 00_config_02_repseqs_.yaml - source_term: dada2pe_max_ee_r - `dada2_trunc_q`: Quality score for truncation - Default: 2 - source_file: 00_config_02_repseqs_.yaml - source_term: dada2_trunc_q - `dada2_pooling_method`: Pooling method for error model - Default: independent - Options: independent | pseudo | pooled - source_file: 00_config_02_repseqs_.yaml - source_term: dada2_pooling_method - `dada2_chimera_method`: Chimeric sequence detection method - Default: consensus - Options: consensus | pooled - source_file: 00_config_02_repseqs_.yaml - source_term: dada2_chimera_method - `dada2_min_fold_parent_over_abundance`: Minimum fold parent over abundance - Default: 1 - source_file: 00_config_02_repseqs_.yaml - source_term: dada2_min_fold_parent_over_abundance - `dada2_n_reads_learn`: Number of reads to learn error model - Default: 1000000 - source_file: 00_config_02_repseqs_.yaml - source_term: dada2_n_reads_learn

Diversity Analysis Settings: - `plot_diversity`: Enable alpha diversity plots - Default: True - Options: True | False - source_file: 00_config_02_repseqs_.yaml - source_term: plot_diversity - `alpha_max_depth`: Maximum depth for alpha diversity plots - Default: 500 - source_file: 00_config_02_repseqs_.yaml - source_term: alpha_max_depth - `core_sampling_depth`: Sampling depth for core diversity metrics - Default: 500 - source_file: 00_config_02_repseqs_.yaml - source_term: core_sampling_depth

Optional Filtering: - `to_filter`: Enable filtering - Default: False - Options: True | False - source_file: 00_config_02_repseqs_.yaml - source_term: to_filter - `repseq_min_length`: Filter sequences by minimum length - Default: 0 (0 = no filtering) - source_file: 00_config_02_repseqs_.yaml - source_term: repseq_min_length - `repseq_max_length`: Filter sequences by maximum length - Default: 0 (0 = no filtering) - source_file: 00_config_02_repseqs_.yaml - source_term: repseq_max_length - `repseq_min_abundance`: Minimum abundance threshold - Default: 0 (0 = no filtering) - Options: [0-1] - source_file: 00_config_02_repseqs_.yaml - source_term: repseq_min_abundance -

`repseq_min_prevalence`: Minimum prevalence threshold - Default: 0 (0 = no filtering) - Options: [0-1] - source_file: 00_config_02_repseqs_.yaml - source_term: repseq_min_prevalence

Important Notes: - Set length limits to extreme values (e.g., 0, 10000) to skip length filtering - Set abundance/prevalence to 0 to skip abundance filtering - Adjust truncation lengths based on your primer sequences and expected amplicon size - Consider your sequencing depth when setting diversity analysis parameters

2.3 Running Step 2 repseqs

There are two main ways to execute Tourmaline for this step:

2.3.1 Command-Line Execution You can run Tourmaline directly from the command line using the `tourmaline.sh` script:

```
./tourmaline.sh --step repseqs \
--configfile 00_config_02_repseqs_<RUN_NAME>.yaml \
--cores <threads>
```

Or using short flags:

```
./tourmaline.sh -s repseqs \
-c 00_config_02_repseqs_<RUN_NAME>.yaml \
-n <threads>
```

- `--step/-s`: The pipeline step to run (here, `repseqs`).
- `--configfile/-c`: Path to the config file for this run.
- `--cores/-n`: Number of threads/cores to use for this run.

Note: The number of threads/cores should match the value set in your config file (e.g., `asv_threads`). If they differ, the command-line value may override the config, or the pipeline may use the value provided in `00_config_02_repseqs_<RUN_NAME>.yaml`.

2.3.2 SLURM Array for Parallel Processing The `sbatch_tourmaline2_step2.sh` script is designed to run the Tourmaline repseqs step in parallel across multiple seq_run_name using a SLURM job array. **Before running this script, you must configure the SLURM script to match your cluster environment and project requirements.** For additional details, see sbatch_tourmaline2_step1.sh script configuration.

```
sbatch sbatch_tourmaline2_step2.sh \
--working-directory <OUTPUT_PARENT_DIR> \
--config-file 00_config_02_repseqs_<RUN_NAME>.yaml
```

- `--working-directory/--wd`: The parent directory containing all batch/data-type output folders.
- `--config-file`: The config file name to use for each sample/batch.

The SLURM script will: - Launch one job per seq_run_name in your working directory (using the SLURM array mechanism). - Automatically set up output logging and skip seq_run_name that already have output. - Activate the correct Conda environment and run Tourmaline for each seq_run_name in parallel.

For SLURM command details, see Monitoring SLURM Jobs and Outputs.

2.4 Step 2 repseqs Outputs

- Inside each seq_run_name folder (in the working directory), step 2 repseqs creates an output folder (e.g., `02_seq_run_name_1_<RUN_NAME>-repseqs/`).
- Inside this output folder, you will find the results of the step 2 repseqs, including processed QIIME2 artifacts and summary files.
- If the output folder already exists, the script will skip reprocessing that sample.

```
WORKINGDIR = /absolute/path/to/user/directory

WORKINGDIR/
    seq_run_name_1/
        seq_run_name_1_Output/
            01_seq_run_name_1_<RUN_NAME>-qaqc/
                ...
            02_seq_run_name_1_<RUN_NAME>-repseqs/
                02_seq_run_name_1_<RUN_NAME>-repseqs.fasta
                02_seq_run_name_1_<RUN_NAME>-repseqs.qza
                02_seq_run_name_1_<RUN_NAME>-table.biom
                02_seq_run_name_1_<RUN_NAME>-table.qza
                02_seq_run_name_1_<RUN_NAME>-table.tsv
                stats/
                    *.txt
                    *.tsv
                    *.qzv
                    *.qza
```

- **Representative sequences** contain the actual DNA sequences of your ASVs:
  - `02_seq_run_name_1_<RUN_NAME>-repseqs.fasta` - FASTA file containing the representative sequences (ASVs)
  - `02_seq_run_name_1_<RUN_NAME>-repseqs.qza` - QIIME2 artifact containing the representative sequences
- **Feature Table** shows how many times each ASV appears in each sample:
  - `02_seq_run_name_1_<RUN_NAME>-table.qza` - QIIME2 artifact containing the ASV table (counts per sample)
  - `02_seq_run_name_1_<RUN_NAME>-table.tsv` - Tab-separated version of the feature table for easy viewing
  - `02_seq_run_name_1_<RUN_NAME>-table.biom` - BIOM format table for compatibility with other tools

The `stats/` folder contains comprehensive quality assessment and statistical outputs. These files help you assess data quality, sample coverage, and diversity patterns:

- **DADA2 Statistics** (dada2_stats.*), if using DADA2 denoising:
  - Denoising statistics showing error rates, merged pairs, and filtering results
  - Quality score distributions and error profiles
- **Feature Table Summaries**:
  - Sample metadata summaries
  - ASV frequency distributions
  - Sample depth statistics
  - Rarefaction curves
- **Alpha Diversity Metrics**:
  - Observed features (richness)
  - Shannon diversity
  - Faith's phylogenetic diversity
  - Evenness metrics
  - Statistical comparisons between groups
- **Visualization Files** provide interactive plots for exploring your data:
  - `.qzv` files - QIIME2 visualization files that can be viewed in the QIIME 2 web browser
  - `.qza` files - QIIME2 artifacts containing the underlying data for visualizations

2.5 Evaluate Step 2 RepSeqs Outputs

After completing the step 2 repseqs, it's important to evaluate the quality of your results. The `02_extract_dada2_stats.py` script provides a comprehensive summary of DADA2 denoising statistics across all your samples.

2.5.1 Script overview The `02_extract_dada2_stats.py` script:

1. Searches recursively through your `working_directory/02_seq_run_name_1_<RUN_NAME>-repseqs/stats/` to find all **`dada2_stats.tsv`** files generated by step 2 repseqs
2. **Extracts key metrics** from each file:
   - Percentage of input passed filter
   - Percentage of input merged (for paired-end data)
   - Percentage of input non-chimeric
3. **Computes statistics** for each metric:
   - Minimum value across samples
   - Maximum value across samples
   - Mean value across samples
4. **Highlights potential issues** by coloring mean values in red if they fall below 70%
5. **Presents results** in a formatted table showing the statistics for each sample directory

2.5.2 Usage

```
WORKINGDIR = /absolute/path/to/user/directory

python tourmaline/scripts/02_extract_dada2_stats.py \
--working-directory $WORKINGDIR
```

2.5.3 Interpreting the results The script generates a table showing statistics for each seq_run_name, making it easy to identify samples with poor denoising performance and assess overall data quality across your entire dataset.

Example table output with 3 seq_run_name:

```
+-------------------------------------+------------------+------------------+------------------+-----
| seq_run_name    | percentage of input passed filter (Min%) | percentage of input passed filter (Max%)
+-------------------------------------+------------------+------------------+------------------+-----
| seq_run_name_1  | 85.23            | 95.67            | 92.45            | 78.34            | 89.12
| seq_run_name_2  | 72.45            | 88.90            | 81.67            | 65.23            | 82.34
| seq_run_name_3  | **45.67**        | **62.34**        | **54.23**        | **38.90**        | **55.67
+-------------------------------------+------------------+------------------+------------------+-----
```

Interpretation: - **seq_run_name_1**: Good quality with high percentages across all metrics - **seq_run_name_2**: Acceptable quality but some metrics below 70% (non-chimeric mean) - **seq_run_name_3**: Poor quality with all metrics below 70% and indicate potential issues: - Poor sequencing quality - Primer mismatches - Sample contamination - Insufficient read depth

Recommendations for poor quality samples:

If you encounter samples like seq_run_name_3 with consistently low percentages, consider adjusting the following parameters in your `00_config_02_repseqs_<RUN_NAME>.yaml` file (see 00_config_02_repseqs_.yaml Configuration for field details):

To improve "passed filter" percentage: - Increase `dada2_max_ee_f` and `dada2pe_max_ee_r` (e.g., from 2 to 3-5) to allow more reads with higher error rates - Decrease `dada2_trunc_len_f` and `dada2pe_trunc_len_r` to retain more of the read length - Adjust `dada2_trim_left_f` and `dada2pe_trim_left_r` to skip problematic regions at read starts

To improve "merged" percentage: - Decrease `dada2_trunc_len_f` and `dada2pe_trunc_len_r` to increase overlap between forward and reverse reads - Ensure your primer sequences are correctly specified in the Step 1 config file

To reduce chimera percentage: - Increase `dada2_min_fold_parent_over_abundance` (e.g., from 1 to 2-4) for stricter chimera detection - Consider using `dada2_chimera_method: "consensus"` (default) or try `"pooled"` for more aggressive chimera removal

General quality improvements: - Increase `dada2_n_reads_learn` (e.g., to 2000000) to improve error model learning - Consider using `dada2_pooling_method: "pseudo"` instead of `"independent"` for better error modeling

Notes: After making changes, re-run the step 2 repseqs and re-evaluate the statistics using `02_extract_dada2_stats.py` script. Monitor the trade-off between read retention and quality - more permissive settings may retain more reads but could introduce more errors. If you don't want to overwrite previous results but keep a history of your runs, consider changing the `run_name` field in `00_config_02_repseqs_<RUN_NAME>.yaml` to create a new output folder (e.g., `02_seq_run_name_1_<UPDATE_RUN_NAME>-repseqs/`).

3 STEP 3: TAXONOMY

3.1 Step 3 (TAXONOMY) overview

The taxonomy step is responsible for assigning taxonomic classifications to your ASVs using various classification methods. The main actions performed in this step are:

- **Taxonomic classification:** The pipeline supports multiple classification methods
  - **Naive-Bayes**: Machine learning-based classification using pre-trained classifiers
  - **Consensus BLAST**: BLAST-based classification with consensus assignment
  - **Consensus VSEARCH**: VSEARCH-based classification with consensus assignment
  - **Bowtie2-BLCA**: Bowtie2 alignment followed by BLCA classification
- **Database handling:** Automatic import and conversion of reference databases (FASTA, QZA formats)
- **Taxonomic summarization:** Creates taxonomic tables at specified taxonomic levels
- **Visualization:** Generates interactive taxonomic barplots and summary statistics
- **Output formats:** Produces QIIME2 artifacts (`.qza`), visualizations (`.qzv`), and exportable formats (`.tsv`, `.biom`)

Notes: The pipeline automatically handles input from the previous RepSeqs step and applies the appropriate classification method based on your configuration.

3.2 `00_config_03_taxonomy_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME>.yaml` Configuration

The `00_setup_tourmaline2_analyses.py` script automatically populates the `00_config_03_taxonomy_<CLASSIFIER_METHO` file with common settings for all seq_run_name analyses. However, you should review all fields based on your specific data characteristics and research objectives before running the pipeline.

3.2.1 Key Configuration Fields

Basic Settings: - `run_name`: Automatically set to `03_<seq_run_name>_<CLASSIFIER_` by the setup script - `output_dir`: Automatically set to the absolute path of your output directory - `sample_metadata_file`: Path to your metadata file (automatically set) - `repseqs_qza_file`: Path to the representative sequences from Step 2 (automatically set) - `table_qza_file`: Path to the feature table from Step 2 (automatically set) - `classify_threads`: Number of threads for classification (set via `--threads` argument)

Classification Method Selection: - `classify_method`: Choose from: - `naive-bayes`: Machine learning-based classification (recommended for well-curated databases) - `consensus-blast`: BLAST-based classification with consensus assignment - `consensus-vsearch`: VSEARCH-based classification with consensus assignment - `bt2-blca`: Bowtie2 alignment followed by BLCA classification

Database Configuration: - `database_name`: Name of the reference database (automatically extracted from database path) - `refseqs_file`: Path to reference sequences (FASTA or QZA format) - `taxa_file`: Path to reference taxonomy (TSV or QZA format) - `pretrained_classifier`: Path to pre-trained classifier (for naive-bayes method)

Classification Parameters: - `taxa_ranks`: Taxonomic ranks in database (default: kingdom,phylum,class,order,family,genus
- `collapse_taxalevel`: Taxonomic level for summary table collapse (default: 7)

Method-Specific Parameters:

Naive Bayes: - `skl_confidence`: Confidence threshold for classification (default: 0.7)

Consensus BLAST/VSEARCH: - `perc_identity`: Percentage identity threshold (default: 0.8) - `query_cov`: Query coverage threshold (default: 0.8) - `min_consensus`: Minimum consensus threshold (default: 0.51)

Bowtie2-BLCA: - `bowtie_database`: Path to Bowtie2 database (optional) - `confidence_thres`: Confidence threshold (default: 0.8)

Notes: For naive-bayes, provide a pre-trained classifier and for consensus methods, reference sequences and taxonomy files are required.

3.3 Running Step 3 taxonomy

There are two main ways to execute Tourmaline for this step:

3.3.1 Command-Line Execution You can run Tourmaline directly from the command line using the `tourmaline.sh` script:

```
./tourmaline.sh --step taxonomy \
--configfile 00_config_03_taxonomy_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME>.yaml \
--cores <threads>
```

- `--step/-s`: The pipeline step to run (here, `taxonomy`).
- `--configfile/-c`: Path to the config file for this run.
- `--cores/-n`: Number of threads/cores to use for this run.

Notes: The number of threads/cores should match the value set in your config file (e.g., `classify_threads`). If they differ, the command-line value may override the config, or the pipeline may use the value provided in `00_config_03_taxonomy_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME>.yaml`.

3.3.2 SLURM Array for Parallel Processing The `sbatch_tourmaline2_step3.sh` script is designed to run the Tourmaline taxonomy step in parallel across multiple seq_run_name using a SLURM job array. **Before running this script, you must configure the SLURM script to match your cluster environment and project requirements.** For additional details, see sbatch_tourmaline2_step1.sh script configuration.

```
sbatch sbatch_tourmaline2_step3.sh \
--working-directory <OUTPUT_PARENT_DIR> \
--config-file 00_config_03_taxonomy_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME>.yaml
```

- `--working-directory/--wd`: The parent directory containing all batch/data-type output folders.
- `--config-file`: The config file name to use for each sample/batch.

The SLURM script will: - Launch one job per seq_run_name in your working directory (using the SLURM array mechanism). - Automatically set up output logging and skip seq_run_name that already have output.
- Activate the correct Conda environment and run Tourmaline for each seq_run_name in parallel.

For SLURM command details, see Monitoring SLURM Jobs and Outputs.

3.4 Step 3 taxonomy Outputs

- Inside each seq_run_name folder (in the working directory), step 3 taxonomy creates an output folder (e.g., `03_seq_run_name_1_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME>-taxonomy/`).
- Inside this output folder, you will find the results of the step 3 taxonomy, including processed QIIME2 artifacts and summary files.
- If the output folder already exists, the script will skip reprocessing that sample.

```
WORKINGDIR = /absolute/path/to/user/directory

WORKINGDIR/
    seq_run_name_1/
        seq_run_name_1_Output/
            01_seq_run_name_1_<RUN_NAME>-qaqc/
                ...
            02_seq_run_name_1_<RUN_NAME>-repseqs/
                ...
            03_seq_run_name_1_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME>-taxonomy/
                03_seq_run_name_1_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME>-taxonomy.qza
                03_seq_run_name_1_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME>-taxonomy.tsv
                03_seq_run_name_1_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME>-taxa_sample_table_l7.ts
                03_seq_run_name_1_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME>-asv_taxa_features.tsv
                figures/
                    03_seq_run_name_1_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME>-taxa_barplot.qzv
```

- **Taxonomic classifications** contain the assigned taxonomy for each ASV:
  - `03_seq_run_name_1_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME>-taxonomy.qza`: QIIME2 artifact containing taxonomic classifications
  - `03_seq_run_name_1_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME>-taxonomy.tsv`: Tab-separated version of taxonomic classifications
- **Taxonomic summary tables** show abundance at different taxonomic levels:
  - `03_seq_run_name_1_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME>-taxa_sample_table_l7.tsv`: Collapsed table at specified taxonomic level
  - `03_seq_run_name_1_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME>-asv_taxa_features.tsv`: ASV sequences with taxonomic assignments

The `figures/` folder contains interactive visualizations for exploring your taxonomic data: - `03_seq_run_name_1_<CLASSIFIE`
Interactive taxonomic barplot that can be viewed in the QIIME 2 web browser

5-PREPARING INPUTS FOR ANALYSES SUBMISSION TO OCEAN DNA EXPLORER

After completing all three Tourmaline steps, you may need to prepare your analysis metadata for submission to the Ocean DNA Explorer (ODE). The `format_analysisMetadata.py` script helps you generate standardized metadata files from your Tourmaline configuration files.

1 Scripts overview

1.1 Individual Analysis: `format_analysisMetadata.py`

The `format_analysisMetadata.py` script:

1. **Reads configuration files** from all three Tourmaline steps:
   - Step 1: QA/QC configuration (`00_config_01_qaqc_<RUN_NAME>.yaml`)
   - Step 2: RepSeqs configuration (`00_config_02_repseqs_<RUN_NAME>.yaml`)
   - Step 3: Taxonomy configuration (`00_config_03_taxonomy_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME`

2. **Extracts and formats metadata** into standardized terms including:
   - **FAIR eDNA terms**: Standardized metadata fields for environmental DNA analysis
   - **Custom terms**: Tourmaline-specific parameters and settings
   - **Software versions**: QIIME2, DADA2, Deblur, and other tool versions
3. **Generates a TSV file** with all metadata terms and their values for easy submission to the Ocean DNA Explorer (ODE)

1.2 Batch Processing: `generate_ODE_analysis_inputs.py`

The `generate_ODE_analysis_inputs.py` script automates the metadata generation process across multiple analyses:

1. **Searches recursively** through your working directory to find all analysis folders
2. **Automatically identifies** the latest configuration files for each step:
   - Latest QA/QC config file
   - Latest RepSeqs config file
   - Latest taxonomy config file matching your specified classifier and database
3. **Generates metadata** for each analysis using `format_analysisMetadata.py`
4. **Optionally copies ODE input files** (feature tables and ASV-taxonomy files) to an ouptut folder
5. **Handles batch processing** of multiple analyses with consistent project metadata

2 Usage

2.1 Individual Analysis: `format_analysisMetadata.py`

```
python format_analysisMetadata.py \
  -s 00_config_01_qaqc_<RUN_NAME>.yaml \
  -r 00_config_02_repseqs_<RUN_NAME>.yaml \
  -t 00_config_03_taxonomy_<CLASSIFIER_METHOD>_<DATABASE_NAME>_<RUN_NAME>.yaml \
  -p <PROJECT_ID> \
  -a <ASSAY_NAME> \
  -A <ANALYSIS_RUN_NAME> \
  -o <OUTPUT_FILE>.tsv
```

Required Arguments: - `-s, --samples_config`: Path to Step 1 QA/QC config file - `-r, --repseqs_config`: Path to Step 2 RepSeqs config file - `-t, --taxonomy_config`: Path to Step 3 Taxonomy config file - `-p, --project_id`: Project identifier for data submission - `-o, --output`: Output TSV file path

Optional Arguments: - `-a, --assay_name`: Assay name (defaults to amplicon_name from samples config) - `-A, --analysis_run_name`: Analysis run name (defaults to run_name from taxonomy config) - `-T, --tourmaline_metadata`: Path to tourmaline metadata file (default: `./00-data/tourmaline_metadata.yaml`)

2.2 Batch Processing: `generate_ODE_analysis_inputs.py`

```
python tourmaline/scripts/generate_ODE_analysis_inputs.py \
  --working-directory <WORKING_DIRECTORY> \
  --classify-method <CLASSIFIER_METHOD> \
  --database <DATABASE_NAME> \
  --project-id <PROJECT_ID> \
  --assay-name <ASSAY_NAME> \
  --output-folder-path <OUTPUT_FOLDER> \
  --ode-inputs TRUE
```

Required Arguments: - `--working-directory, --wd`: Base directory containing all analysis folders - `--classify-method`: Classification method used (consensus-blast, naive-bayes, consensus-vsearch, bt2-blca)

- `--database`: Database name used for taxonomy assignment - `--project-id`: Project identifier for data submission - `--assay-name`: Assay name (e.g., COI, 12S, 18S)

Optional Arguments: - `--output-folder-path, -O`: Output folder for metadata files and ODE inputs (if not provided, files are saved in each analysis folder) - `--ode-inputs`: Whether to copy ODE input files (TRUE/FALSE, default: FALSE)

3 Script outputs

Both scripts generate a tab-separated values (TSV) file with two columns: - `term_name`: The metadata field name - `values`: The corresponding value for that field

3.1 Key term names

FAIR eDNA Terms: - `project_id`: Your project identifier - `assay_name`: Name of the assay/amplicon - `analysis_run_name`: Name of this analysis run - `sop_bioinformatics`: Standard operating procedure reference - `trim_method`: Software used for trimming - `trim_param`: Trimming parameters and approach - `merge_tool`: Software used for read merging - `error_rate_tool`: Software used for error rate estimation - `chimera_check_method`: Chimera detection method - `otu_clust_tool`: ASV/OTU clustering tool - `otu_db`: Reference database used - `tax_asign_cat`: Taxonomy assignment category - `tax_class_id_cutoff`: Identity cutoff for taxonomy assignment

Custom Tourmaline Terms: - `qiime2_version`: QIIME2 version used - `tourmaline_asv_method`: ASV method (DADA2, Deblur) - `dada2_trunc_len_f/r`: DADA2 truncation lengths - `dada2_max_ee_f/r`: DADA2 maximum expected errors - `skl_confidence`: Naive Bayes confidence threshold - `min_consensus`: Consensus threshold for BLAST/VSEARCH - `confidence_threshold`: BLCA confidence threshold

3.2 ODE Input Files

When using `generate_ODE_analysis_inputs.py` with `--ode-inputs TRUE`, the script automatically copies the following files to your output folder:

1. **Feature Table** (`02_*-table.tsv`):
   - Source: Latest RepSeqs output folder
   - Content: ASV abundance table with sample counts
   - Processing: Header row is automatically removed for ODE submission
2. **ASV-Taxonomy Features** (`03_*asv_taxa_features.tsv`):
   - Source: Latest taxonomy output folder
   - Content: ASV sequences with taxonomic assignments
   - Format: Tab-separated file with ASV ID, sequence, and taxonomy

Notes: These files are essential for ODE submission.

GLOSSARY

1 seq__run__name

A unique identifier for each sequencing run, batch, or sample.

2 PROTOCOL INFORMATION

2.1 Minimum Information about an Omics Protocol (MIOP)

- MIOP terms are listed in the YAML frontmatter of this page.
- See https://github.com/BeBOP-OBON/miop/blob/main/model/schema/terms.yaml for list and definitions.

2.2 Authors

PREPARED BY	AFFILIATION	ORCID	DATE
Luke Thompson	NOAA/AOML, MSU/NGI	https://orcid.org/0000-0002-3911-1280	2024-12-06

- All authors known to have contributed to the preparation of this protocol should be listed, including those who filled in the template.
- Visit https://orcid.org/ to register for an ORCID.

2.3 Protocol Revision Record

VERSION	RELEASE DATE	DESCRIPTION OF REVISIONS
1.0.0	2025-12-09	Initial release
1.0.1	2025-12-15	Updated YAML front matter

- Version numbers start at 1.0.0 when the protocol is first completed and will increase when changes that impact the outcome of the procedure are made (patches: 1.0.1; minor changes: 1.1.0; major changes: 2.0.0).
- Release date is the date when a given protocol version was finalised.
- Description of revisions includes a brief description of what was changed relative to the previous version.

NEED TO EDIT HEADINGS BELOW…

3 RELATED PROTOCOLS IN YOUR FOLDER

This is a list of other protocols deposited in your folder which should be known to users of this protocol. For example, if you create a derivative or altered protocol, you would link to the original protocol in the section below. Please include the link to each related protocol. Also include the version number of that protocol when you linked to it.

PROTOCOL NAME AND LINK	VERSION The version of the protocol you linked to	RELEASE DATE This is the date corresponding to the version listed to the left
Content Cell	Content Cell	yyyy-mm-dd
Content Cell	Content Cell	yyyy-mm-dd

4 RELATED EXTERNAL PROTOCOLS

This is a list of other protocols that are not in your folder which should be known to users of this protocol. These include, e.g., kit manuals. Please upload all relevant external protocols to Appendix A and link to them here.

EXTERNAL PROTOCOL NAME AND LINK	ISSUER / AUTHOR Please note who authored the protocol (this may also be a company name)	ACCESS DATE This is the date you downloaded or scanned the protocol and uploaded it.
Content Cell	Content Cell	yyyy-mm-dd
Content Cell	Content Cell	yyyy-mm-dd

4.1 Acronyms and Abbreviations

ACRONYM / ABBREVIATION	DEFINITION
NOAA	National Oceanographic and Atmospheric Administration
AOML	Atlantic Oceanographic and Meteorological Laboratory
MSU	Mississippi State University
NGI	Northern Gulf Institute

5 GLOSSARY

SPECIALISED TERM	DEFINITION
Content Cell	Content Cell
Content Cell	Content Cell

EQUIPMENT, SOFTWARE & PACKAGES

NAME	VERSION OR MODEL	MANUFACTURER OR CREATOR	REMARKS
Equipment			
e.g. Laptop	Content Cell	Content Cell	e.g. needs at least 16 GB of RAM
Content Cell	Content Cell	Content Cell	Content Cell
Software			
Content Cell	Content Cell	Content Cell	Content Cell
Content Cell	Content Cell	Content Cell	Content Cell
Code			
Please include the links to the code you used for this analysis			
e.g. link to the released version of a github repository	Content Cell	Content Cell	Content Cell
Content Cell	Content Cell	Content Cell	Content Cell

1 GUIDE TO ARCHIVED METHODOLOGY

The contents of this archive should allow your analysis to be reproduced exactly as you intended it.

This document provides guidance on the contents of each partner's compressed archive of in-silico methods. This document should be part of that same archive, serving as an extended README.

Below, please find guidance on what this archive should include. When describing the contents of the archive, please give precise file names and relative paths to the files.

2 Archive content

To reproduce the in-silico analysis, please provide one of the following (in order of decreasing preference)

1. Jupyter, R notebook(s) or equivalents
2. Downloaded archive of (the released version of) your github repository

3. Individual scripts

In each of the above cases, guidance and documentation for all the steps you took to perform the in-silico analysis should be included. In case 1., code and documentation are integrated. In cases 2. and 3., in-line comments may be provided, however, these are not generally sufficient as documentation. In those cases, please provide a step-by-step protocol on how and when to run each script in the Execution Procedure section below.

Please include a script on **data acquisition** (e.g. documentation and code to pull sequences from INSDC, access sequences on an institutional FTP server, download metadata files, check file integrity via md5 checksum). Please add sufficient detail, so that the partners only have to install the software, run this script and will then have all the data needed to perform any analysis described below.

2.1 Code

Here please describe each file containing code, including its purpose, its input, its output. Please provide the names and the relative paths to this documentation.

2.2 Code documentation

Here, please indicate if your documentation is with the code (in a code notebook) or stored separately. In-line comments are not considered documentation. If the documentation is stored separately, please provide the names and the relative paths to this documentation.

2.3 Metadata

Please provide link(s) to the files containing metadata about your sequence data (e.g. environmental data, procedural data). Please see the MIxS compliant metadata guidance.

Auxiliary files e.g. mapping files, test/dummy files, colour palette

3 Execution Procedure

Please fill out this section if you have not already documented it as part of your R, Jupyter, or similar notebook. In this section, please provide a step-by-step guidance on how and when to run each component of your code.

4 Quality control

In this section please include the names and paths that can be used to validate that operations were successful. If such checks were done during the execution procedures, please note this here. We recommend identifying such steps with in-line tags (e.g. “#QC”).

5 Basic troubleshooting guide

Identify known issues associated with the procedure, if any. Provide troubleshooting guidelines when available.

REFERENCES

Insert all references cited in the document. Please insert full DOI address when available, e.g. http://doi.dx.org/10.1007/s1125
014-0404-1

APPENDIX A: DATASHEETS

Link to any documents such as software guidelines, images, etc that support this protocol. Please include a short note describing the document's relevance.

UNDER CONSTRUCTION

Example output directory structure:

```
OKEX
  16S
    SEQRUN1
        okex-16Ssr1-dt-qaqc #discard untrimmed
          fastq.qza
          qaqc_config.yaml
          raw_fastq.qza
          stats/
        okex-16Ssr1-dt-da2peTa-repseqs #discard untrimmed, dada2PE, trunc option a
          repseqs_config.yaml
          okex-sr1-dt-da2peTa-repseqs.fasta
          okex-sr1-dt-da2peTa-repseqs.qza
          okex-sr1-dt-da2peTa-table.biom
          okex-sr1-dt-da2peTa-table.qza
          okex-sr1-dt-da2peTa-table.tsv
          stats/
        okex-16Ssr1-dt-da2peTa-nbS2501-taxonomy #discard untrimmed, dada2PE, trunc option a, naive-baye
          okex-sr1-dt-da2peTa-nbS2501-asv_taxa_features.tsv
          okex-sr1-dt-da2peTa-nbS2501-taxa_sample_table_l7.tsv
          taxonomy_config.yaml
          okex-sr1-dt-da2peTa-nbS2501-taxonomy.qza
          okex-sr1-dt-da2peTa-nbS2501-taxonomy.tsv
          classifier.qza
          figures/
        okex-16Ssr1-dt-da2peTa-cbS2501-taxonomy #discard untrimmed, dada2PE, trunc option a, consensus
          ...

    SEQRUN2
      ...
  18S
...
```