Applying the FAIR principles and reproducibility in Pipeline Development.

Thomas Vogel, Franziska Sassen, Jan Steiger, Bhavya Sree Kandi

October 25, 2024

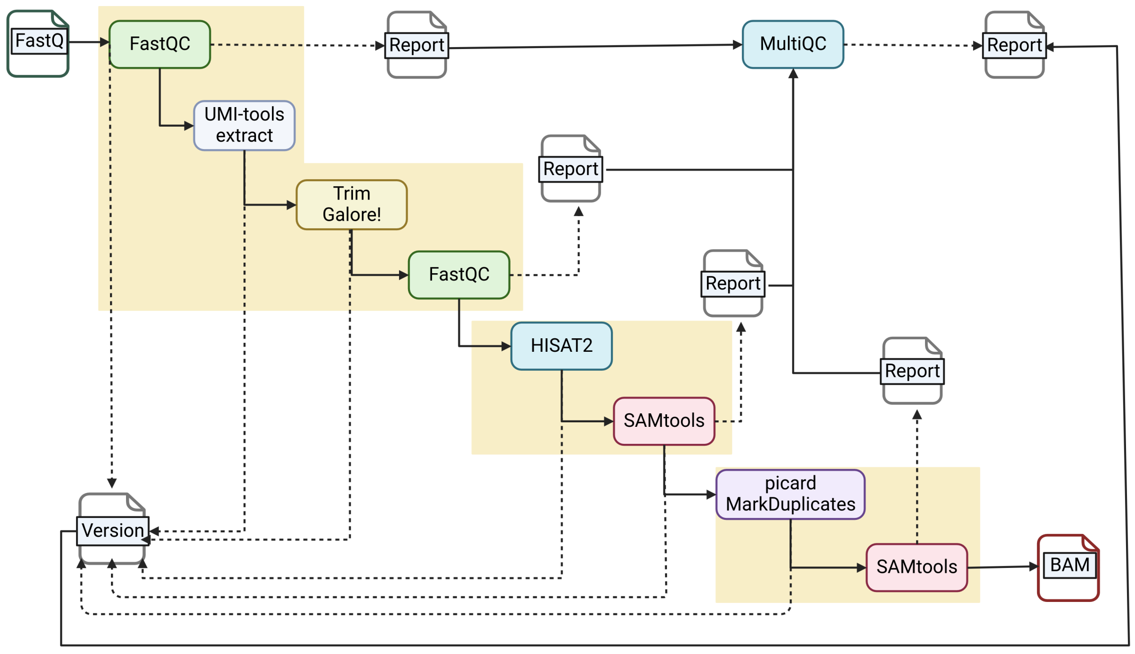

Figure 1: Flowchart presenting the steps of the pipeline.

1 Introduction

Advances in RNA sequencing (RNA-seq) have transformed transcriptomics by enabling comprehensive analysis of gene expression. However, the complexity of RNA-seq data analysis requires computational pipelines that automate key steps from quality control to differential expression analysis, ensuring reproducibility and scalability. By standardizing workflows, pipelines make it easier to handle large datasets, reducing human error and delivering consistent, reliable and interpretable results for various biological and clinical applications. [6]
Reproducibility is a keystone of scientific research, further enhanced by adhering to the FAIR principles. The *nf-core* framework adheres to the FAIR principles, ensuring that pipelines are Findable, Accessible, Interoperable, and Reusable for reproducible bioinformatics analyses. Each pipeline is openly available on `GitHub` under the MIT license, making it accessible and easily findable, with detailed documentation on installation and usage. Interoperability is ensured through containerization (i.e. Docker, Singularity) and compatibility with various platforms, from laptops to High performance compute clusters (HPCs) and cloud servers. Rigorous continuous integration (CI) testing and stable `GitHub` releases guarantee that pipelines are well-tested and reliable, enhancing their reusability across diverse environments.
nf-core pipelines are thoroughly tested and peer-reviewed, ensuring they adhere to best practices and are maintained by a large community of bioinformaticians. This makes *nf-core* pipelines reliable for large-scale data processing.[3]

2 Methods

2.1 *nf-core* pipeline framework

The community surrounding *nf-core* provides rigorous guidelines for pipeline design, ensuring the reproducibility and scalability of analysis workflows in accordance with the FAIR principles. For this, the main pipeline script plays a pivotal role, as it initializes and validates input parameters before invoking workflows from the `workflows/` and `subworkflows/` directory. The primary responsibility of these workflows is the organization of primary and modular analysis tasks, which often integrate additional reusable components within the `modules/` directory. This allows for seamless integration and adaptation across different workflows.
Pipeline configuration is centralized through the `nextflow.config` file, which sets global parameters and adapts the pipeline to various computational environments via predefined profiles. Additional configuration files can be found in the `conf/` directory, which define further customization, such as resource allocation, to ensure accessibility across different systems. The `nextflow_schema.json` file enforces consistency and validation of input parameters through a structured `JSON` schema, facilitating findability and interoperability by enabling automated tools to understand and utilize the requirements of a pipeline.
Accessibility is a key consideration, with comprehensive documentation available in the `README.md` file and the `docs/` directory. This includes usage instructions, details on implemented methods, and data format descriptions. These resources facilitate the ease of locating and comprehending the deployment of the pipeline and interpretation of its results, thereby supporting both accessibility and reusability. Supporting files such as `LICENSE` and `CITATIONS.md` ensure the pipeline is open and properly credited, fostering an environment of collaboration and reuse. The `CHANGELOG.md` maintains a transparent history of updates, aiding in the findability of changes and ensuring users are informed of the latest developments.
Moving forward, the *nf-core* framework shall serve as guidance for building an analysis pipeline that aligns with the FAIR principles and establishes a clear organizational structure, modular design, and detailed documentation—key elements

2.2 Software and computational environment

To implement the presented analysis pipeline, `Conda` (v24.7.1) was utilized for environment and package management, thereby streamlining the installation and maintenance of dependencies. Specifically, it was used to install `Nextflow` (v23.10.1) as the workflow management system and `nf-core` (v2.11.1), which provides standardized and community-curated workflow components for analyses. `Docker` (v24.0.5) and `Singularity` (v3.8.6) were used to enable the containerization of aforementioned components, ensuring consistent execution environments across different systems. The repository for this pipeline is available on GitHub at: `https://github.com/COMATOS3/WF_Seminar_Pipeline`.
The analysis was conducted on a machine equipped with an `AMD Ryzen 7 5800X` processor and 32 GB of DDR4 RAM operating at 3800 MHz. To show the performance of the pipeline under the best conditions available to us, the pipeline was run using sixteen CPU cores and 30 GB of RAM.

3 Results

3.1 Building the pipeline

To gain hands-on experience in pipeline design using `Nextflow`, and to internalize the concepts of reproducibility and modularity, we build our own pipeline for RNA quality control and alignment. A general flowchart of the pipeline can be found at the beginning of the report (Figure 1). The pipeline has four main functions: the quality of the reads is assessed using `FastQC`, reads are trimmed using `Trim Galore!`, aligned using `HISAT2` and then duplicates are identified using `Picard`. To gain insight into the processes and their results, statistics are reported several times during the pipeline, both form the used software itself and using `SAMtools`, and `MultiQC` is used to generate a final report at the end, which also contains the versions of each software used.
The pipeline was build using the organisation into different subworkflows exhibited by published *nf-core* pipelines as a guide. The internal organisation of the pipeline is described in the following paragraphs, a detailed flowchart can be found in the Appendix (section 5).
It begins with sample input and initialization, using the **nf-core/validation** plugin. The input for

this process is the samplesheet, which contains file paths for the input sequencing data, as well as metadata that describes the samples. The reads are parsed and categorized as either single-end or paired-end. Two outputs are generated, a channel containing the metadata and the corresponding read files (**reads_ch**), as well as a log summarizing the input parameters.
The channels containing the reads and metadata, as well as relevant parameters, then functions as the input for the first subworkflow, named **FASTQ_FASTQC_UMITOOLS_TRIMGALORE**. Here, the module **FASTQC** assesses the quality of the raw reads and the trimmed reads, which are generated by the next two processes, the module **UMITOOLS_EXTRACT** extracts Unique Molecular Identifiers (UMIs) and the module **TRIMGALORE** trims adapters and low-quality bases from the reads, as well as filtering out based on the number of reads remaining after the trimming. All three of these processes can be skipped if desired, by using the appropriate flags when running the pipeline. This subworkflow outputs six different channels: **ch_filtered_reads** contains the trimmed and filtered reads, which are to be used in the alignment. Channels **ch_fastqc_raw_multiqc** and **ch_fastqc_trim_multiqc** contain the reports generated by **FASTQC**. Channels **ch_trim_log_multiqc**, **ch_trim_read_counts** contain the trimming logs and read counts, respectively, and **ch_versions** contains the software versions used in this subworkflow.
Before the reads can be aligned, the genome that they are to be aligned to needs to be locally available, preprocessed and indexed. To ensure this, the **PREPARE_GENOME** subworkflow, which we based on the equivalent workflow in *nf-core/rnaseq* [5] is used. It takes genome files as input, these can be `FASTA`, `GFF` or `GTF` files, as well as splice site information and a `HISAT2` index directory. If necessary, files are downloaded from available sources, preprocessed and indexed. The subworkflow has five output channels: **ch_hisat2_index**, **ch_splicesites**, **ch_fasta** and **ch_fai** contain the prepared genome data that will be used in the alignment, and **ch_versions**, which is updated with the genome preparation tool versions. Read alignment is performed in the **FASTQ_ALIGN_HISAT2** subworkflow. It takes the genome data containing channels from **PREPARE_GENOME** and the channel containing the filtered reads from **FASTQ_FASTQC_UMITOOLS_TRIMGALORE** as input, and contains two processes: in **HISAT2_ALIGN** reads are aligned to the reference genome, and in **BAM_SORT_STATS_SAMTOOLS** the generated `BAM` files are sorted and alignment statistics are generated. This subworkflow has seven output channels: **ch_versions** is updated with the tools used in this subworkflow, **ch_genome_bam** and **ch_genome_bam_index** contain the aligned `BAM` files and their indexes, **ch_samtools_stats**, **ch_samtools_flagstat** and **ch_samtools_idxstats** contain the alignment statistics and **ch_hisat2_multiqc** contains a summary of the `HISAT2` runs, to be used in `MultiQC`.
After alignment, the last analytic step is to mark duplicate reads. This happens in the **BAM_MARKDUPLICATES_PICARD** subworkflow, and can be skipped, either automatically when UMIs are extracted or by using the appropriate flag when running the pipeline. The subworkflow takes the genome files generated by **PREPARE_GENOME** as well as the aligned `BAM` files from the previous step as input. It performs three processes, **PICARD_MARKDUPLICATES** identifies and marks duplicate reads, **SAMTOOLS_INDEX** indexes the deduplicated `BAM` files and **BAM_STATS_SAMTOOLS** generates statistics for the duplicate free `BAM` files. This subworkflow updates the channel containing the versions of all tools, as well as the channels containing statistics generated during alignment. It also generated duplicate free `BAM` files and indexes as well as metrics as outputs.
To improve the readability of the results of the pipeline and for documentation, two subworkflows are used. **CUSTOM_DUMPSOFTWAREVERSIONS** collects the versions of each tool that is used from **ch_versions** and writes them into a `YAML` file. **MULTIQC** collects the quality metrics, software versions and workflow summaries generated by each process, and generates a comprehensive report in the form of an `HTML` file.

3.2 Running the pipeline

The pipeline can be run from the terminal, using the command

```
nextflow run ./rna-bfjt.nf \
--input samplesheet.csv \
--outdir <OUTDIR> \
--genome <GENOME> \
--profile <docker/singularity>
```

when in the directory, and with the packages, described in section 2, installed. Further parameters

and their usage can be found on the project's `GitHub` page.
Several configuration files are used to configure different aspects of the pipeline. `nextflow.config` is used to manage global and default parameters and can be edited by the user, for example to adapt to hardware specifications. `conf/base.config` manages resource allocations based on max values and process labels, it is based on *nf-core*'s `base.config`. `conf/genomes.config` manages the paths and resource identifiers to attributes of different genomes and is based on *nf-core*'s `igenome.config`; `HISAT2` index and `GTF` for example species need to be manually included here, as they were missing in the original configuration. `conf/publish.config` defines the behaviour of generating output files and is based on *nf-core/rnaseq*'s [5] configuration. `schema_input.json` and `nextflow_schema.json` define and describe parameters for validation and are based on *nf-core/rnaseq*'s [5] configuration. Last but not least, `methods_description_template.yml` and `multiqc_config.yml` manage `MultiQC` configuration and methods description and are also based on *nf-core/rnaseq*'s [5] configuration.

3.3 Testing the pipeline

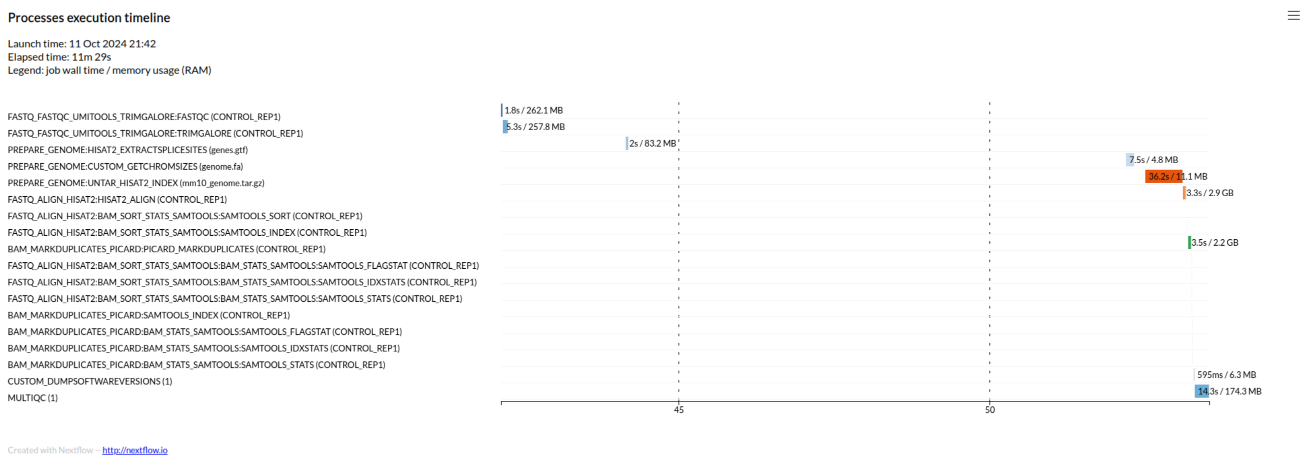

Figure 2: Timeline generated by `Nextflow` of the *Mus musculus* example

To test our pipeline, we ran it on two examples: one minimal example, containing on sample of paired-end reads, each file shortened to 1,200 reads from *Mus musculus*, and one larger example, containing twelve paired end samples, with each file shortened to 50,000 reads, generated in *Saccharomyces cerevisiea*. In both test runs, we used `Nextflow`'s `-with-timeline` function to generate a timeline and review the performance of our pipeline.
In the example run using the mouse sample, the total time taken was 11 minutes and 29 seconds. A significant portion of this time was spent downloading the genome, which caused a noticeable gap in the graph as can be seen in Figure Figure 2. The actual processing tasks, such as genome indexing, were much quicker and took a maximum of only 36 seconds.
The run using *S. cerevisiae* samples was faster despite having more and longer samples, primarily due to its smaller genome size, speeding up both the download and the alignment. This run took only 4 minutes and 20 seconds. Similar to the mouse run, there was a large gap in the timeline during the genome download. The process was significantly sped up by processing files in parallel, which is more efficient compared to running the tools sequentially on each sample, showing the power of `Nextflow`'s automatic parallelization features.
The samplesheets and sample files to run these tests can be found on the project's `GitHub` repository. While in this small examples the rate limiting step is the genome download, it stands to reason that with full RNA sequencing files, the time needed for alignment would increase significantly.

4 Discussion

4.1 FAIR principles in *nf-core*

The goal of `Nextflow` is to enable scalable and reproducible scientific workflows[2], with *nf-core* using `Nextflow` to build analysis pipelines. With the emerging concept of FAIR data usage, *nf-core* has a variety of guidelines and best practices to adhere to FAIR principles and can therefore be

taken as an example what FAIR is about and how to apply it to scientific research. The main features encompass documentation, stable releases, open source, CI testing, run anywhere and packaged software, each of which is alleviating the difficulty of data analysis in science in some way. Apart from that, there are many requirements and some recommendations for guidelines to ensure FAIR usage.[4] These requirements detail the implementation and documentation of the pipelines even further, including many of the FAIR principles explicitly. Some examples of the requirements mapped from the principles are the explicit mention of a DOI for each pipeline, open source availability with MIT license and the usage of a streamlined vocabulary, only to name a few.
The *nf-core* website hosts a repository for each pipeline, with an integrated search function and tags for the pipelines. This makes the individual pipelines very easy to find from the host website. Another part of the findability is the extensive documentation for the pipelines and the DOIs for many of them.
The pipelines have their documentation on the *nf-core* website and are easily accessible through `GitHub`, since they are required to be open source. This makes both the documentation and the pipelines themselves retrievable for everyone.
As already mentioned, the documentations and pipelines have to adhere to specified schemata. One such example is that workflows have to be built using `Nextflow` and using the *nf-core* template. Generally, the community values the usage of a common formatting and language, allowing for interoperability between different workflows.
The software is managed by containers and as such there is no need to install all the tools used by the workflow. This takes of version compatibilities and is continuously updated and tested. In combination with guidelines about the usage of pipelines, *nf-core* workflows are easy to run on most machines and have a high degree of reusability.
Working with scientific data, there is often a lack of application of FAIR principles, both because of the relatively new emergence of the principles and difficulty in altering storage of already existing data.[1] As a young community, *nf-core* took the ideas of FAIR research software to heart and tied them to their required standard for published pipelines. Therefore, we observe a strict adherence to those ideas, making the workflows in their repository an exceptional example of FAIR software.
For our pipelines, we could not adhere to all of the guidelines and requirements, since our project was not as extensive as the regular pipelines. While we did not use the *nf-core* template, we used the same basic structure for the workflow as in other pipelines. Since our project is not meant to be published as it is a proof-of-concept, it is especially lacking in the findability department. It will not be added to the repository on the *nf-core* website and thus not publicly available. This ties into the accessibility of our pipeline. On a similar note, our pipelines are not meant to be used by others and so we have not as extensive of a documentation as the *nf-core* pipelines. As our pipeline is an isolated project, there will be no updates and it will degrade over time. Lastly, there is no CI Testing in our project and only limited testing in general.
In summary, most of the limits regarding FAIRness of our workflow are a result of the project design. We implemented the vocabulary and processes according to community best practices, applying some of the FAIR principles in the process, as well as documenting usage and implementation of the workflow, but because of the framework and timeline there is no room to include all of them as properly as other *nf-core* pipelines.

4.2 Evaluation of our workflow

To determine, whether our pipeline constitutes a strong RNA-seq analysis workflow, we compared it to the *nf-core/rnaseq* pipeline.
The most important thing, that was not included in our pipeline, is the assembly and quantification. Therefore, our pipeline does not resolve in a `BED` file but rather ends with the `SAM` file. This should be the first priority when extending the pipeline, since it is the most important part missing for more in-depth analysis.
After that, there are several options one could add to improve the quality of the analysis. For that, there are tools to exclude genome contaminants and ribosomal RNA. Additionally, our pipeline only allows for `HISAT2` as alignment, because of computational limitations, and no pseudoalignment, but further steps could include other alignments.
Lastly, the addition of some quality of life changes can make the pipeline more user friendly. In the *nf-core* pipeline, there are some extra steps in the preprocessing, like inferring the strandedness, and a final quality control.
Altogether, our pipeline is a strong skeleton for an RNA-seq analysis workflow, which can be extended upon. To use it for real analysis, at least the transcript assembly and quantification

should be added.

References

[1] Esther Thea Inau et al. “Initiatives, Concepts, and Implementation Practices of FAIR (Findable, Accessible, Interoperable, and Reusable) Data Principles in Health Data Stewardship Practice: Protocol for a Scoping Review”. In: *JMIR research protocols* 10 (2 Feb. 2021). ISSN: 1929-0748. DOI: `10.2196/22505`. URL: `https://pubmed.ncbi.nlm.nih.gov/33528373/`.

[2] *nextflow home*. Accessed: 11 October 2024. URL: `https://www.nextflow.io`.

[3] *nf-core — nf-co.re*. `https://nf-co.re`. [Accessed 15-10-2024].

[4] *nf-core Requirements and Recommendations*. Accessed: 11 October 2024. URL: `https://nf-co.re/docs/guidelines/pipelines/overview`.

[5] Harshil Patel et al. *nf-core/rnaseq: nf-core/rnaseq v3.16.0 - Further Fire Ferret*. Version 3.16.1. Oct. 2024. DOI: `10.5281/zenodo.13940293`. URL: `https://doi.org/10.5281/zenodo.13940293`.

[6] *RNA Sequencing and Analysis — ncbi.nlm.nih.gov*. `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4863231/`. [Accessed 15-10-2024].

5 Appendix

5.1 Diagram of the pipeline

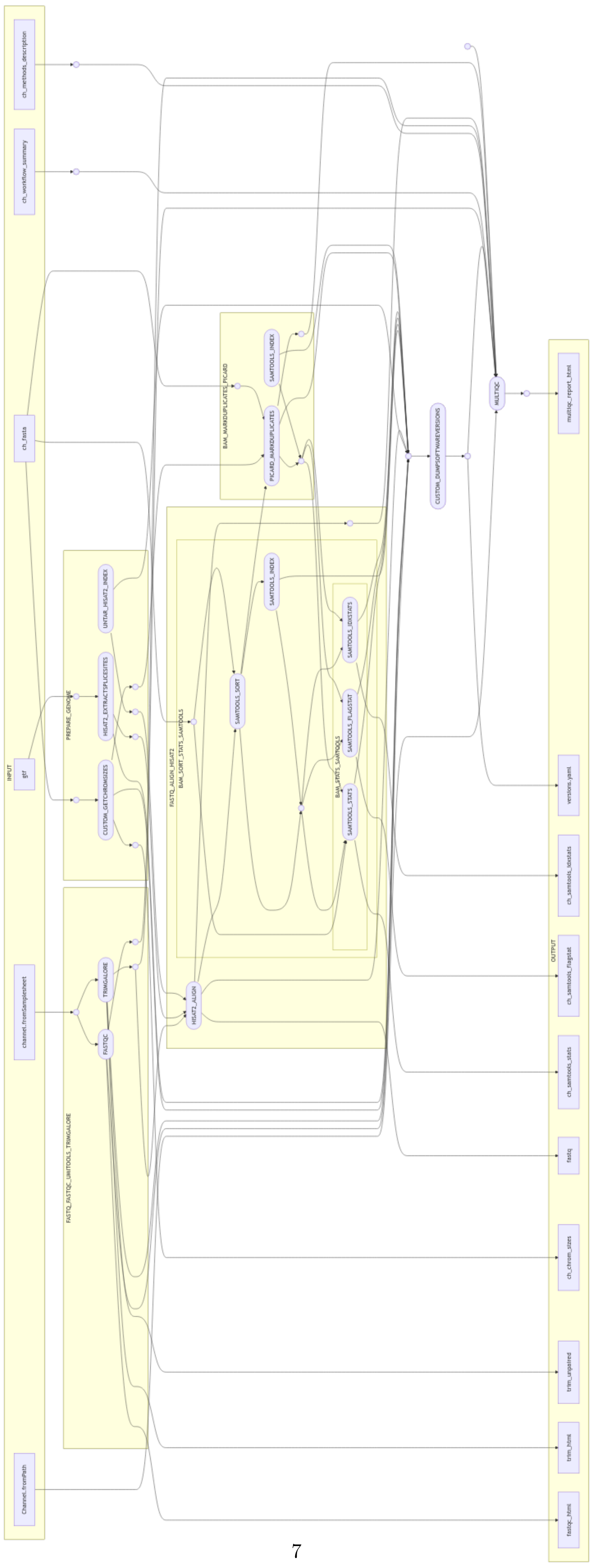

Figure 3: Diagram of the pipeline, its subworkflows and modules