Simple RNA Sequencing Pipeline: A Nextflow Implementation

Tabea Attig mat.nr:6563671, Weronika Jaśkowiak mat.nr:6761084, Annika Maier mat.nr:6366281, Maike Nägele mat.nr:5474675

***Abstract*—The increasing availability of RNA-sequencing (RNA-seq) data has created a demand for efficient and reproducible computational workflows. This project presents a modular RNA-seq analysis pipeline developed using Nextflow, a powerful workflow management tool. The pipeline automates the process of RNA-seq data analysis, incorporating essential tools such as HISAT2, STAR, FastQC, Trim Galore!, Picard and SAMtools. It efficiently handles raw sequencing data, performs quality control, read trimming and alignment, while ensuring reproducibility through the use of containerization and version control.**

I. Introduction

Biomedical research has advanced over the years due to the rapid development of high-throughput technologies. Scientists around the world are conducting experiments and collecting complex data using established guidelines and well-developed pipelines. In science, it is crucial to ensure the reproducibility of experimental procedures and computational analyses.

In the field of computational biology, ensuring reproducibility is particularly challenging. In response, workflow managers were developed. Additionally, the nf-core community was established to maintain developed workflows. Every nf-core pipeline is written in Nextflow, allowing them to run on a wide range of computational infrastructures and integrate with container technologies such as Docker and Singularity. The nf-core community designed a special framework that enables scientists to reliably reproduce pipelines in every institute [1].

Even minor changes in software versions, settings, or reference annotations can significantly affect results. Workflow managers address this issue by automating the tracking of input parameters and software tool versions within computational pipelines. This leads to improved consistency and reliability in research results. The nf-core community also encourages collaboration among researchers by providing a platform for sharing and improving pipelines. This joint effort speeds up innovation and ensures that the workflows are continuously updated. The nf-core community continues to expand and adapt each year to meet new challenges and innovations [2].

In particular, RNA sequencing (RNA-seq) has been a vital technique in biomedical research [3]. It is an advanced sequencing method that allows for the comprehensive analysis of a biological sample's transcriptome and to uncover novel RNA species. The process starts with RNA extraction, which is then converted into complementary DNA and prepared for sequencing. This technique provides the quantification of gene expression levels, the discovery of novel transcripts and the identification of alternative splicing patterns. [4]. The RNA-seq is widely recognized as the gold standard for quantifying gene expression across the entire transcriptome, both in research and clinical settings [5].

Template credits: M. Shell was with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332 USA e-mail: (see http://www.michaelshell.org/contact.html).

II. Goal of the project

The aim of our mini-project was to develop a reproducible, Simple RNA Sequencing Pipeline, consisting of checking the quality of the data, trimming raw reads, performing alignment and lastly marking the duplicates.

III. Materials and Methods

To develop our Simple RNA Sequencing Pipeline, we considered six steps: validation of given by user sample sheet, quality assessment of raw reads, removal adapters and enhancement of the quality of the data, alignment, sorting alignment output and marking duplicates. As a reference, we looked at the already developed RNA-seq pipeline [6].

A. Nf-core pipeline general structure

The general structure of the nf-core pipeline can be broken down into four main components: `main.nf`, which executes the entire pipeline; `modules`, which contains specific processes for executing tools or scripts; `subworkflow`, which is a set of modules that work together toward a common purpose; and `nextflow.config`, which includes parameters and different types of profiles, such as execution profiles [7].

B. Reference nf-core/rna-seq pipeline structure

The reference nf-core/rna-seq pipeline consists of five stages: pre-processing, genome alignment and quantification, pseudo-alignment and quantification, post-processing and final quality control. In the first stage, the pipeline incorporates tools such as FastP, FastQC, Trim Galore!, UMI-tools and others. Specifically, the subworkflow `FASTQ_FASTQC_UMITOOLS_TRIMGALORE` handles quality control, UMI extraction and trimming of RNA-seq reads. It uses several modules, including `FASTQC`, `UMITOOLS_EXTRACT` and `TRIMGALORE`, to perform these tasks. The next steps involve alignment and quantification, followed by pseudo-alignment and quantification. The former includes tools such as STAR, HISAT2 and SALMON,

while the latter relies solely on SALMON for quantification. In the fourth stage, files are sorted and indexed using tools such as SAMtools. The subworkflow `BAM_SORT_STATS_SAMTOOLS` has been integrated to execute those tasks. Additionally, duplicate marking is performed using picard MarkDuplicates. The final stage generates comprehensive quality reports using tools such as DESeq2, Preseq and others [6].

C. Testing data

For testing, the nf-core/rnaseq pipeline includes a test dataset available on GitHub [2]. The data we used originates from the yeast species *Saccharomyces cerevisiae*, consisting of 101bp paired-end, strand-specific RNA reads. This dataset was collected to investigate the mechanisms by which the transcription factor Rap1 regulates noncoding transcription and suppresses divergent transcription at gene promoters in yeast [8].

D. Development environment

To create our pipeline, PyCharm v2023.3.+ [9], VisualStudioCode v1.86.+ [10] and the workflow manager Nextflow v23.10.+ [7] were used.

E. Validation of samplesheet

A short Python script, `validation_samplesheet.py`, was implemented to validate the samplesheet given by the user. It is crucial to ensure that the input is in the correct format, preventing errors during the execution of the pipeline.

F. Quality assessment

For checking the quality of raw reads, FastQC v0.12.1 [11] was used, by pulling the respective Docker container. Using this tool is a good practice for basic quality control checks on raw data [11]. It produces a file (.html) containing the final quality report of the sample and files (.zip) containing respective reports for reverse and forward ends.

G. Adapter removal and enhancement of data quality

To remove adapters and enhance the quality of the data, Trim Galore! v0.6.7 [12] was employed, by pulling the respective Docker container. Additionally, post-trimming quality assessments are produced by the same tool by adding the parameter `--fastqc`. It outputs trimmed single-end and paired-end files (.fq.gz), trimmed details files (.txt) and files (.html) containing final quality reports. This step essentially cleans up the data to ensure working with relevant, high-quality reads.

H. Alignment

To compare sequenced reads to the reference genome and determine the most likely origin of each read, the STAR v2.7.10b [13] and HISAT2 v2.2.1 [14] were employed, by pulling respective Docking containers.

The HISAT2 uses a hierarchical indexing approach that combines the Burrows-Wheeler transform with the FM index to efficiently align RNA-seq reads. It constructs a global FM index for the entire genome, along with several local FM indexes that enable quick alignment extensions. HISAT2 handles reads in a single pass and at the same time gathers splice site information to align both long and short sequences with high sensitivity and speed. The advantage of the HISAT2 tool is that it requires only 4 GB of memory, making it ideal for use on standard desktop computers [14]. It produces a file (.sam), which includes details such as the position of each read on the genome, the quality of the alignment and any mismatches or gaps.

The STAR aligner is an algorithm, written in C++. What makes it different from other aligners is its ability to directly align non-contiguous sequences, thereby detecting both canonical and non-canonical splice junctions, as well as chimeric transcripts. The algorithm consists of two parts: the seed-finding phase to search for a Maximal Mappable Prefix and creates alignments of the entire read sequence by combining all the seeds that were aligned to the genome in the first phase. STAR is capable of fast mapping, processing up to 550 million paired-end reads per hour on modern multi-core servers. It combines speed with accuracy and scalability, making it suitable for analyzing large transcriptome datasets [13]. For small-size files, it can be used on standard desktop computers. STAR aligner produces an unsorted file (.bam) that contains information similar to the HISAT2 output.

I. Sorting alignment output

Before marking duplicates, sorting the file (.sam) is required. The tool SAMtools v1.20 [15] was used for this step, by pulling the respective Docker container. The sorting process arranges the alignment data in a specific order, usually by the position of reads on the reference genome. SAMtools outputs a sorted file (.sam).

J. Marking duplicates

Marking duplicates is essential for further downstream analysis, i.e. for variant calling. Duplicates can arise during sample preparation, such as from PCR during library construction, or from instrument failure. Especially in RNA-seq where duplicate sets are typically large, duplicate marking ensures that each RNA fragment is counted only once and removes biases from duplicate reads. For this step, the MarkDuplicates module of Picard v3.2.0-6 [16] was used, by pulling the respective Docker container. MarkDuplicates compares sequences in a file (.bam) or (.sam) to identify and rank duplicates based on their base-quality. The tool outputs a new file (.bam), which contains detailed information about each read, indicating if it's flagged as a duplicate, along with other relevant data [16]. It also creates a metrics file detailing the number of duplicate reads.

IV. Results

A. Pipeline

The resulting pipeline can be seen in Fig. 1. It consists of six tools and modules. In order to ensure modularity, each

tool is called in its own module. The pipeline is intended to be run from the `main.nf` file. The pipeline creates a separate folder within the same directory, organizing the results of each tool into individual subfolders. The pipeline can be executed on any container specified using the profile flag, ensuring reproducible results regardless of the device used. However, we recommend running this pipeline within a Docker container, as it has been tested in that environment. The user can validate the pipeline by running a test on the default test data provided before applying it to their own dataset. It's important to test both aligners by specifying the aligner option (`--align HISAT2/STAR`).

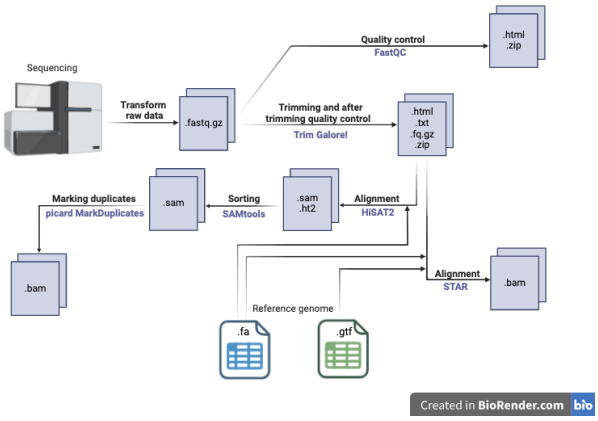

Fig. 1. Outline of the resulting pipeline

To initiate the test, user can execute the following command from the nf-core directory:

```
nextflow run main.nf
    -profile docker
    --align HISAT2/STAR
```

To run the pipeline for their analysis, the user must create a samplesheet that includes the correct paths to the data. Additionally, the user needs to specify the appropriate parameters for their analysis:

```
--samplesheet <PATH TO SAMPLESHEET>
--outdir <OUTDIR>
--fasta <GENOME FASTA>
--gtf <GTF>
--align <STAR>
--length <MINIMAL LENGTH OF TRIMMED READS>
--adapter <ADAPTER SEQUENCE>
```

The nf-core/ directory serves as the root folder of the pipeline, following the standard conventions of nf-core. Inside the root directory, the following components are present:

- `main.nf`: The main Nextflow script of the pipeline orchestrates the flow of data through each step, managing the entire process from raw sequencing reads to fully processed, aligned data. It specifies the order in which modules are executed, including tasks such as quality control, read trimming and alignment with tools like HISAT2 or STAR. Default parameters are defined within the script, allowing users to customize the pipeline via the command line without needing to modify the underlying code,
- `nextflow.config`: The configuration file specifies the parameters, such as max_memory and max_cpu. Additionally, profiles and registry settings for the Nextflow pipeline are defined. Modules reference profiles in the `nextflow.config` file, allowing them to automatically adjust parameters at runtime based on the selected profile,
- `bin/`: It includes an executable Python script, `validation_samplesheet.py`, used to validate the samplesheet. The sample sheet is passed to the `SAMPLESHEET_VALIDATION` process within the `samplesheet_validation` module via a channel. This process ensures the sample sheet, used to process paired-end FASTQ files, adheres to specific criteria by checking the header format, verifying no missing values in any row and confirming that both FASTQ files exist and are correctly named with the .fastq.gz extension. The function outputs a text file stating "Samplesheet validation passed!" if successful, or "Samplesheet validation failed!" with detailed error messages if validation issues are detected,
- `data/`: The pipeline's test input data includes a samplesheet.csv located in the `samplesheets/` directory, which contains both correct and incorrect samplesheets for validation testing. Correct samplesheets include either two or eight samples, while incorrect ones are used to test the validation process. Additionally, the test data includes a reference genome that can be used for alignment, comprising a FASTA file (`genome.fa`) with reference genome sequences and a GTF file (`genome.gtf`) that provides gene location and feature annotations. Paired-end sequencing reads in FASTQ format are also provided for testing the pipeline's alignment and downstream processes,
- `modules/`: Each module contains a `main.nf` file that outlines the processes to be executed in the workflow. These processes utilize a Conda environment defined in the `environment.yml` file, which supplies the required software and dependencies for the module,
  - `samplesheet_validation`: The validity of a sample sheet is checked using the `validate_samplesheet.py` script. The process runs this Python script, captures the output in a file named `validation.txt` and determines whether the validation passed or failed based on the contents of that file. If the validation fails, detailed error messages are both displayed in the console and written to the `validation.txt` file. These messages specify the exact issues found for each sample (missing values, incorrectly named files,...),
  - `FASTQC/`: runs FastQC on the paired-end FASTQ files listed in the sample sheet to evaluate the quality of the sequencing data [11]. The process generates a FastQC report (.html) for each FASTQ file, along with a `versions.yml` file documenting the ver-

sion of FastQC used in the analysis,
  - `trimgalore/`: This process performs read trimming on paired-end FASTQ files using Trim Galore!, with optional adapter and minimum read length trimming based on user-specified parameters [12]. If provided, the `--adapter` option specifies the adapter sequence and `--length` sets the minimum read length for trimming. The process also runs FastQC for quality control, generating trimmed reads, reports and optional output files (.html) and compressed reports. Additionally, it generates a `versions.yml` file that logs the versions of the tools used during the process,
  - `hisat2/align/`: It consists of two processes: `HISAT2_BUILD`, which generates HISAT2 index files (i.e. .1.ht2 to .8.ht2) required for `HISAT2_ALIGN`, which align paired-end FASTA/FASTQ sequencing reads to the reference genome, producing an aligned file (.sam),
  - `STAR/`: This module consists of two processes: `INDEX_FILE` and `STAR_ALIGN`. The `INDEX_FILE` process generates a STAR genome index using a reference genome and GTF file, saving the index in a specified output directory to facilitate downstream alignment steps. The `STAR_ALIGN` process aligns paired-end reads to the indexed genome with STAR, producing sorted BAM files. Additionally, it logs the versions of the tools used in a `versions.yml` file [13],
  - `picard_mark_duplicates/`: Executes duplicates step using Picard,
  - `samtools/`: Defines sorting step executed by SAMtools,
- `results/`: default directory to store output, i.e. `results/fastqc`.

The developed pipeline does not include any subworkflows, as seen in the reference nf-core/rna-seq pipeline in section III-B, due to its low complexity and the number of tools used.

B. Benchmarking

The benchmarking for our RNA-sequencing pipeline was conducted on a machine equipped with 16 GB of RAM, an 8-CPU 64-bit system, running Ubuntu 22.04. For testing, two different sample sizes were used: one paired-end sample and another set with eight paired-end reads. We used paired read samples from a pre-packaged dataset that had been modified to be a subset of the original data to reduce size and computing time for the weaker machine. The specific files used in the first sample were *SRR63570707_1.fastq.gz* and *SRR6357070_2.fastq.gz*. For the lager sample, the fastq files for *SRR6357070* - *SRR6357077* have been used.

The pipeline was run using the following command for the small sample set:

```
nextflow run main.nf
   -profile docker
   --outdir benchmark_test
   -with-report benchmark_results
```

The pipeline was run using the following command for the larger sample set:

```
nextflow run main.nf
   -profile docker
   --outdir benchmark_test_8_files
   --samplesheet 'data/samplesheets/
       samplesheet_8_samples.csv'
   -with-report
       benchmark_results_8_samples
```

The default path of the samplesheet and the reference genome, as well as the tool HISAT2 for alignment, were used. The total runtime for this process using the default files was **22.9 seconds**. During this run, processes such as `SAMPLESHEET_VALIDATION`, `FASTQC`, `TRIMMING` and `HISAT2_BUILD` were able to execute in parallel since they did not depend on each other. However, subsequent processes like `HISAT2_ALIGN`, `SAMTOOLS_SORT_AND_INDEX` and `PICARD_MARK_DUPLICATES` depended on the output of the preceding processes, preventing them from benefiting from Nextflow's parallel processing capabilities. As illustrated in Figure 2, the TRIMMING step took the longest, which would have been a bottleneck in a traditional pipeline.

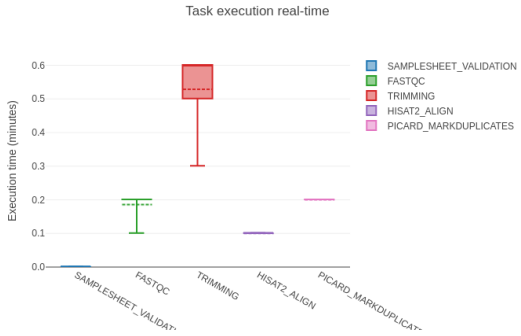

Fig. 2. Task execution time of the eight sample process

When running the pipeline manually with the same settings, the process took around **14.8 seconds**, about 10 seconds less than the Nextflow run. This difference can be attributed to Nextflow's overhead in setting up environments for each process. However, if we account for the time spent manually configuring the environment in the manual run, Nextflow would have likely been faster.

For further assessment additionally, the slightly larger dataset of 8 paired reads has been used. The samplesheet `samplesheet_8_samples.csv` has been used along the standard reference genome, as well as the tool HISAT2 for alignment. The total runtime for this process using these settings was **59.7 seconds**. Similarly to the smaller run, the processes `SAMPLESHEET_VALIDATION`, `FASTQC`, `TRIMMING` and `HISAT2_BUILD` were able to execute in parallel. The larger amount of samples highlighted the parallelization more clearly.

When running the pipeline via the script `benchmarking.sh` with the same settings, including optional parameters to suppress the console output, the process took **181.17 seconds**, about 80 seconds more than the Nextflow run. Here the benefits of the parallelization become more clear and are likely to expand with an increase in samples.

From this, we conclude that setting up a Nextflow pipeline may not be necessary for running one or two files but shows significant advantages when handling large datasets or when the pipeline is run multiple times on a machine that can leverage Nextflow's multi-threading capabilities.

V. Discussion

A. Reproducibility and FAIR Pipelines with nf-core

The nf-core framework plays a crucial role in achieving reproducibility and compliance with FAIR (Findable, Accessible, Interoperable, Reusable) principles by incorporating containerization and strict version control. These features ensure that workflows can be executed in consistent environments across different platforms. In our RNA-seq pipeline, we incorporated the aspects of accessibility and reusability. The Nextflow configuration file specifies different profiles. One can set the package management system and container through the profile flag in the command line. The `README` file in the GitHub repository ensures findablity through a proper documentation. Also, results follow a modular organization into subfolders. Each tools stores its results in a folder named by the specific tool.

However, several areas require further refinement to fully align with the FAIR principles. Implementing tests to verify reproducibility, particularly across diverse datasets, would significantly enhance the reliability of the workflow. Moreover, introducing more robust early-stage error detection mechanisms for input files, such as reference genome files, would provide additional safeguards against data processing errors. Comprehensive validation of key input files, like the sample sheet, could prevent issues before execution. To fully meet the FAIR guidelines, integrating metadata standards and generating standardized outputs would be crucial steps, further enhancing the pipeline's reusability and adaptability across different research contexts.

B. Outlook

While the current pipeline is functional, well-documented and user-friendly, several enhancements could transform it into a more comprehensive and robust RNA-seq workflow. One immediate improvement would be expanding the available parameters for each individual tool, allowing for greater customization and flexibility. Furthermore, incorporating alternative tools for trimming and alignment in future iterations could increase the versatility of the workflow, accommodating a broader range of use cases and research needs.

A critical area for future development lies in the realm of testing and validation. Implementing more rigorous error-handling mechanisms, particularly for file validation and edge-case scenarios, would significantly enhance the pipeline's robustness. This would ensure smooth execution across a wider variety of inputs, minimizing the likelihood of failures due to incomplete or corrupted data. Additionally, conducting more comprehensive tests on powerful infrastructure, such as high-performance servers, would further reinforce the pipeline's reproducibility and scalability.

The modular design of the nf-core/rnaseq pipeline readily lends itself to the integration of additional functionalities. Features such as pseudoalignment, transcript quantification post-alignment with tools like SALMON or RSEM and more extensive post-processing steps could be seamlessly incorporated. By doing so, the pipeline would evolve into a more complex and capable RNA-seq data analysis workflow, covering a broader spectrum of the sequencing analysis pipeline.

Some current limitations, particularly those related to computational resources on local devices, could be mitigated by running the workflow on high-performance infrastructures such as the de.NBI Cloud [17]. This would allow for comprehensive benchmarking of our pipeline against the nf-core/rnaseq workflow, focusing on the tools common to both pipelines. However, the scope of these improvements was constrained by the available time and further extensions were limited by the computational power at our disposal.

VI. Conclusion

In this project, we developed a modular and reproducible RNA-sequencing pipeline using the Nextflow framework. The pipeline incorporates key tools such as HISAT2, STAR, FastQC, Trim Galore!, Picard and SAMtools, ensuring efficient and automated data processing from raw sequencing reads to aligned data. By adhering to FAIR principles, we leveraged containerization and strict version control to enhance reproducibility and portability. The benchmarking demonstrated the pipeline's efficiency, especially in managing large datasets, with parallelized processes improving runtime performance.

While the pipeline meets basic RNA-seq workflow needs, there are several areas for improvement to make it more robust and versatile. These include integrating more flexible parameterization, improving error handling and expanding downstream analysis options, such as differential expression analysis. Implementing additional testing and validation procedures would further enhance reproducibility and ensure broader applicability across different datasets.

In conclusion, our pipeline provides a strong foundation for RNA-seq analysis, offering efficiency, modularity and reproducibility. Future enhancements can build on this framework to increase functionality, making the workflow more adaptable to a wide range of sequencing projects.

REFERENCES

[1] W. A. . G. J. Wratten, L., "Reproducible, scalable, and shareable analysis pipelines with bioinformatics workflow managers," *Nature Methods*, vol. 18, no. 10, pp. 1161–1168, 2021.

[2] P. Ewels, A. Peltzer, S. Fillinger, H. Patel, J. Alneberg, A. Wilm, M. U. Garcia, P. Di Tommaso, and S. Nahnsen, "The nf-core framework for community-curated bioinformatics pipelines.," May 2022.

[3] S. R. Ji F, "Rna-seq: Basic bioinformatics analysis," *Current Protocols in Molecular Biology*, vol. 124, no. 1, p. e68, 2018.

[4] M. S. Kukurba KR, "Rna sequencing and analysis," *Cold Spring Harbor Protocols*, vol. 2015, no. 11, pp. 951–969, 2015.

[5] L. M. M. J. e. a. Everaert, C., "Benchmarking of rna-sequencing analysis workflows using whole-transcriptome rt-qpcr expression data," *Scientific Reports*, vol. 7, no. 1, p. 1559, 2017.

[6] H. Patel, P. Ewels, J. Manning, M. U. Garcia, A. Peltzer, R. Hammarén, O. Botvinnik, A. Talbot, G. Sturm, nf-core bot, M. Zepper, D. Moreno, P. Vemuri, M. Binzer-Panchal, silviamorins, L. Pantano, R. Syme, G. Kelly, F. Hanssen, and C. Mertes, "nf-core/rnaseq: nf-core/rnaseq v3.16.0 - fire ferret (3.16.0)." https://doi.org/10.5281/zenodo.13882081, 2024. Zenodo.

[7] C. M. F. E. e. a. Di Tommaso, P., "Nextflow enables reproducible computational workflows," *Nature Biotechnology*, vol. 35, no. 4, pp. 316–319, 2017.

[8] C. M. M. F. F. D. S. A. v. W. F. Wu ACK, Patel H, "Repression of divergent noncoding transcription by a sequence-specific transcription factor," *Mol Cell*, 12 2018.

[9] https://www.jetbrains.com/pycharm/.

[10] https://code.visualstudio.com/.

[11] S. Andrews, F. Krueger, A. Segonds-Pichon, L. Biggins, C. Krueger, and S. Wingett, "FastQC." Babraham Institute, 2012.

[12] S. Andrews, F. Krueger, A. Segonds-Pichon, L. Biggins, C. Krueger, and S. Wingett, "Trim Galore!." Babraham Institute, 2012.

[13] F. S. J. D. C. Z. S. J. P. B. M. C. T. R. G. Alexander Dobin, Carrie A. Davis, "Star: ultrafast universal rna-seq aligner," *Bioinformatics*, vol. 29, no. 1, pp. 15–21, 2013.

[14] L. B. . S. S. Kim, D., "Hisat: a fast spliced aligner with low memory requirements," *Nature Methods*, vol. 12, no. 4, pp. 357–360, 2015.

[15] P. Danecek, J. K. Bonfield, J. Liddle, J. Marshall, V. Ohan, M. O. Pollard, A. Whitwham, T. Keane, S. A. McCarthy, R. M. Davies, and H. Li, "Twelve years of SAMtools and BCFtools," *GigaScience*, vol. 10, 02 2021. giab008.

[16] "Picard toolkit." https://broadinstitute.github.io/picard/, 2019.

[17] "De.NBI cloud." https://cloud.denbi.de/. Accessed: 2024-10-14.