PEGASUS – A Hybrid Genome Assembly Software Using Nextflow Pipelines

James Lubkowitz[1], Julie Dragon[2,3], Emily Curd[2]
Faculty Advisor: Ed Harcourt[1]

1 St. Lawrence University, NY, USA
2 Vermont Biomedical Research Network, University of Vermont, VT, USA
3 Department of Microbiology & Molecular Genetics, University of Vermont, VT, USA

Abstract

Understanding an organism's genetic composition and biology without a reference genome is difficult. For this reason, genome assembly is a critical step in enhancing our grasp of biological systems. This knowledge is pivotal in deciphering how complex phenotypes, including those associated with conditions such as cancer, vary across populations. Identifying genetic markers associated with diseases or distinct populations has the potential for early identification and treatment as well as enhancing our understanding of genetic variation and diversity across species.

Recent advances in sequencing technology, particularly the ability to generate longer fragments of DNA, have allowed for improved analysis and more effective bioinformatic workflows. Integrating short-read and long-read sequencing approaches presents an opportunity to achieve superior genome quality and coverage quickly while maintaining cost efficiency.

We present a comprehensive bioinformatics pipeline developed using Nextflow, a programming language designed for containerized and parallelized software tools, to assemble genomes using next-generation sequencing results.

With the massive computational power required, this pipeline was developed for implementation on a high-performance computing cluster. Furthermore, it was parallelized for accelerated processing and containerized in Singularity to ensure the reproducibility of results across varying environments and machines. Nextflow's modular design enables independent containerization of each process, enhancing pipeline flexibility and adaptability for diverse genomics research.

We developed and tested our pipeline using short- and long-read sequences of the South American Wandering Spider and Brown Bullhead Catfish from the Vermont Biomedical Research Network at the UVM Larner College of Medicine. Using these systems as models, we describe the first reference genomes for *Cupiennius salei* and *Ameiurus nebulosus* which highlights PEGASUS's potential to advance genomic research in any system.

1 Introduction

Building genomes is a fundamental aspect of modern biological research with broad applications across medicine, agriculture, and conservation. A genome is comprised of all the DNA sequences within an organism, made up of millions of chemical bases represented by the letters A, T, C, and G, according to their chemical properties. On a basic level, these sequences are transcribed into RNA which is then translated into proteins. These proteins then perform tasks in the cell that result in biological behavior in the organism [28]. A genome can be used to analyze which genes are present and thus make inferences about the physical attributes of an individual or population. Genomic analysis is used across industries for early disease identification, increase crop production, and insights into genetic diversity [1][2][3].

Genome assembly is the process of reconstructing the complete DNA sequences of an organism using smaller fragments [4]. Genomes vary greatly in size and complexity from relatively small bacterial genomes to much larger plant genomes. Historically, large, and complex genomes such as plants were harder to assemble due to sequencing technology and computational limits. In recent years, sequencing technologies have advanced significantly allowing for longer reads of DNA to be sequenced. In plants, which generally have large genomes, this advancement has resulted in an increase in the number of genomes assembled and an increased quality of assembly to prior methods [5]. Previous assembly techniques relied on short reads. Short reads, typically 50-300 base pairs in length, are very accurate but contain a limited span of genetic sequence that do not resolve complex regions of the genome. On the other hand, long reads, which can span thousands of base pairs, are fairly new technology and have the potential to increase the quality and efficiency of genome assembly. Despite having a higher error rate than short reads, long reads can span longer regions of DNA and thus be particularly useful for assembling complex genomes and identifying structural variations in addition to resolving repetitive regions [6].

As genomes can be quite large the computational resources needed to work with and build a genome is vast. High-performance computing (HPC) has become integral to genome assembly due to the massive amount of data generated [46]. HPC architectures enable efficient processing of sequencing data and running complex assembly algorithms with access to increased computational resources to deliver timely results. By leveraging HPC resources, researchers can tackle larger datasets and more intricate assembly challenges, thereby accelerating the field of genomics.

We introduce PEGASUS, a Pretty Epic Genome Assembly Software Using Sequences, as an all-in-one, user-friendly genome assembly tool that implements hybrid assembly methods (i.e., using both long and short read sequences) to produce reliable results. In addition to genome assembly, PEGASUS analyzes raw reads to gather statistics, and employs various methods to ensure the quality of the assembled genome allowing for a comprehensive process. PEGASUS also provides a reference genome scaffolding option, which allows for high-quality genome assemblies across multiple individuals without requiring extensive read depth, if a suitable reference genome is available. This approach yields significant advantages in terms of reducing sequencing costs, assembly runtime, and computational power needed when building many genomes of the same species.

2 Background

Genome assembly requires an understanding of the biological structure of DNA, how it is sequenced, and the technology implemented to work with genetic reads. Furthermore, knowledge of the high-performance computing clusters necessary to execute PEGASUS; Singularity, the containerization platform used for PEGASUS; and Nextflow, the language utilized to program PEGASUS is crucial for thorough analysis of this software.

2.1 An Introduction to DNA Sequencing Techniques

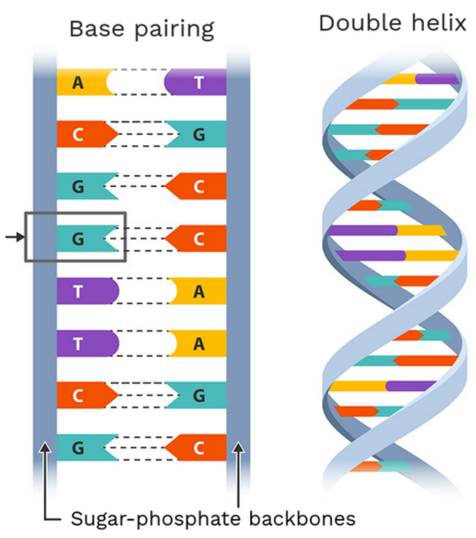


Figure 1. DNA double helix base pairing [7]

Since DNA was discovered, the techniques and technology for analyzing and working with genetic material have advanced dramatically. The first generation of sequencing technology, known as Maxam-Gilbert sequencing, relied on radioactive labeling of DNA fragments and their passage through a gel to determine the position of each base and thus a short sequence. Sanger sequencing, another first-generation technique, was similar but used fluorescence tagging instead of the previous radioactive labeling. In Sanger sequencing fluorescently tagged nucleotides were bound to the template strand and the florescence given off after running through a gel was used to determine the sequence. Much like Maxam-Gilbert sequencing this required DNA to be broken into small fragments of a couple hundred bases to be sequenced at a time, but Sanger's technique was easier to perform and more accurate [8].

Second-generation sequencing saw a massive increase in scalability due to technological advancements in sequencers. Much like first-generation sequencing this required DNA to be broken into small fragments of a couple hundred bases. Unlike first generation sequencing which could only be performed on a single fragment at a time, second generation sequencing allowed for many reads to be sequenced at once. Second generation base identification was done using two enzymes, ATP sulfurylase and Luciferase, that, together, release varying amounts of luminescence as nucleotides were added to the DNA strand. This allowed each base to be determined in live time by the amount of luminescence emitted. This process eliminated the need to run a gel for each fragment of DNA and as many fragments could be sequenced simultaneously second-generation sequencing was extremely scalable. The second-generation sequencing techniques also had the benefit of sequencing both sides of the DNA (as DNA has a double-stranded helix structure). This results in two sequences for each DNA fragment, one in each direction, though they should be complementary (as A and T pair and C and G pair as seen in Figure 1). This results in two raw files containing reads in one direction and reads in the other direction [8].

The first two generations of sequencing technology required DNA to be amplified to ensure there was enough genetic material for sequencers to detect. This required implementing DNA amplification methods such as Polymerase Chain Reaction (PCR) to increase the amount of genetic material in the sample. Third-generation sequencing does not require these additional prep steps before sequencing (though can be and often is performed to increase the total number of reads). This means that an individual strand of DNA can be sequenced with varying success. In addition, maybe even more importantly, third-generation sequencing techniques, such as by Oxford Nanopore, result in sequences of thousands of base pairs, magnitudes larger than previous techniques which consisted of a little over a hundred bases. Oxford Nanopore runs a strand of DNA through a pore that has voltage applied. As the different bases pass through, the ionic flow changes, allowing for the identification of the sequence [8].

To achieve these high through-put sequencing techniques small synthetic DNA sequences, adapters, are bound to the ends of sequences. These adapters are used as binding sites to attach fragments to the sequencing platform and can be used to amplify or identify certain fragments. As the DNA fragments are sequenced these adapter bases are also read and output in the results. These must be removed from the beginning and end of each sequence during the genome assembly process [8].

2.2 Assessing Assembly Quality

For genome quality and analysis purposes it is important to analyze the raw reads prior to assembly. Metrics include the number of duplicate reads, the median length of reads and total number of reads. Further metrics such as the total number of bases is also computed. As the raw reads are prepped for assembly metrics are taken on each step such as the number of reads filtered out or changes before and after adapter removal. In addition to basic read length measurements, raw read's quality scores (Q-scores) are analyzed. Q-scores are calculated for each read during sequencing. Higher Q-scores indicate lower probability of an error in the read. A Q score of 10 corresponds to 10% probability of an incorrect base call while a Q score of 30 corresponds to a 0.1% probability [9]. Many of the metrics used to assess raw reads are also used to assess intermediate and finished genomes.

When assembling a genome, in particular of an organism that has not been sequenced before, assessing the quality of the assembly can be difficult [15]. Various metrics are used to denote a "quality" assembly by comparing the assembled genome to other references such as a previously assembled genomes or the input reads. Implementing multiple quality measurements enables a comprehensive and validated genome assembly [35].

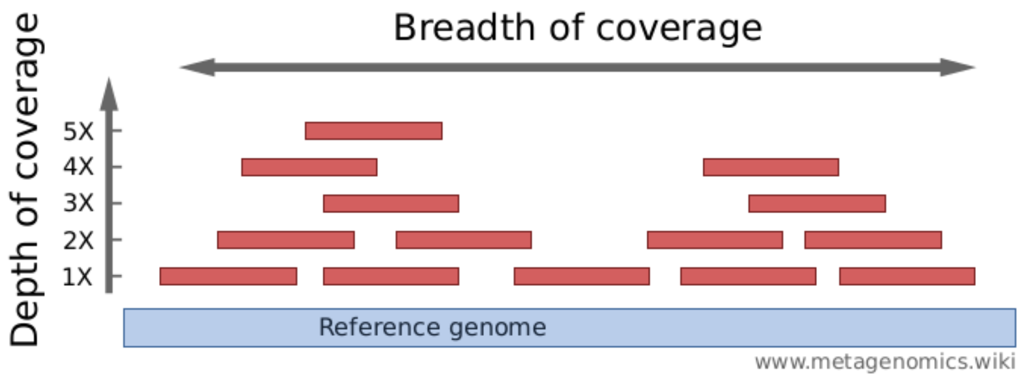


Figure 2. Depth of coverage with reads (in red) aligned on reference genome (in blue) [10].

Depth coverage is an important metric in post genome assembly to assess if sufficient reads were present to produce a quality genome. Depth coverage aligns each fragment of input sequences to the corresponding section of the assembled genome. Depth coverage is then measured as the average number of input sequences across the entire genome at every base. As depth coverage increases, our confidence in each base increases and thus the quality of the genome as a whole [12]. For instance, at 1x coverage—where there is one DNA fragment for every region of the genome—it is challenging to discern if any bases are incorrect, as there is no reference genome or other fragments for comparison. However, at 20x coverage—where there are 20 DNA fragments for each region of the genome—the genome can be assembled with greater certainty. If one read shows an A and the other 19 display a C, an error that would have gone undetected in 1x coverage can be corrected. Because of this read depth is an extremely important metric when assembling genomes to ensure there is ample samples to account for errors in sequencing [11].

While depth coverage is an important assessment of genome quality other metrics such as Busco compare the assembled genome to a reference. Busco uses sequences, single copy orthologs, that are common across related groups (such as order, family or genus). These sequences are universal across all members of the specified group and thus should be present in the assembled genome. For instance, in fish, there might be 1,000 conserved sequences shared across all fish genomes. If an assembled fish genome contains only 400 of these sequences, it can indicate a lower-quality assembly. Busco allows for the selection of different groups depending on the organism being assembled. Busco ratings are displayed as percentages by total complete, fragmented, and missing orthologs [13][14].

Other quality metrics can be employed if a reference genome is available, allowing for faster and more accurate assembly [16]. By comparing the assembled genome sequences with the reference genome and assessing GC content (the percentage of GC bases in the genome), the degree of similarity between the two can be quantified [15]. By aligning the reads of the assembled genome with corresponding reads of the reference genome, commonly referred to as mapping, errors in the assembly can be identified [15][47]. This can also indicate regions of variation between the two genomes that is not a mis-assembly error but rather genetic differences such as structural variants [48].

In addition to comparing the assembled genome to the input reads or reference sequences, analyzing the genome itself can indicate the quality of the assembly. The stretches of sequences within the assembled genome are referred as “contigs”. The length of the longest contig and distribution of contig lengths can provide insights into the assembly. If the assembly process was unsuccessful in generating longer sequences (ie actually assembling a genome from the reads) this could indicate a poor assembly. It's important to note that longer contig lengths do not guarantee a superior assembly quality hence why a variety of assembly metrics are used [35]. Another metric of contig sizes is N50. N50 is the length of the contig at which 50% of the assembled genome is covered by all sequences larger than the value. For this reason, larger N50 scores are better as this indicates that half of the genome is covered by larger contigs. Similar to N50, L50 is the number of contigs that are larger than the N50 value [17]. A genome assembly with an N50 of 36,000 and L50 of 400 would indicate that there are 400 contigs bigger then 36,000 bases that together cover 50% of the genome.

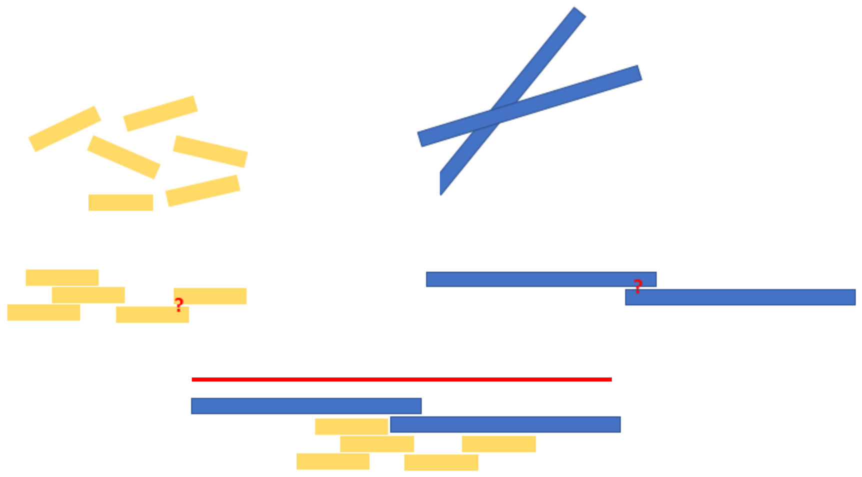


Figure 3. Hybrid scaffolding demonstrating how short reads (in yellow) and long reads (in blue) can be aligned to generate a longer continuous sequence of DNA (in red) [26].

Scaffolding can be done to increase the length of contigs and thus the N50 score. Scaffolding can be used to fill the gaps between contigs and create a longer continues sequences [50]. In addition to increasing the length of contigs, scaffolding techniques allow some insight into the order and orientation of the contig within the genome [51]. The effectiveness of scaffolding depends on the quality and type of the reads being used to scaffold the contigs. As short reads are only a couple hundred bases, using them to scaffold is much less effective and reliable then long read scaffolding [51]. While long reads are effective for scaffolding, the high error rate in long reads can introduce error in this process [51]. To elongate contigs without compromising assembly quality, scaffolding can be followed by additional techniques such as polishing to prevent mis-assembly.

2.3 High-Performance Computing Cluster

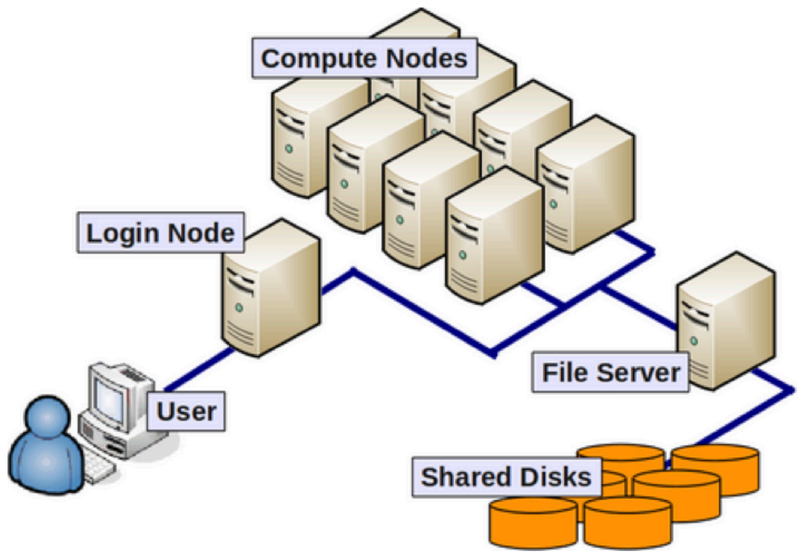


Figure 4. HPC architecture demonstrating remote user access to compute nodes and data storage via shared disks [18].

Genome assembly requires a large amount of genetic sequence data to build a complete genome. This process requires a significant amount of computational power to process and analyze raw DNA sequences which typically surpasses the capabilities computers. Even with an HPC cluster, genome assembly can take several days to compute. To address these challenges, PEGASUS was developed on an HPC Cluster at the University of Vermont to leverage their ability to parallelize tasks and efficiently manage large files.

High-performance computing clusters are large groups of processors that can process data faster than desktops or laptops. These machines have opened the door for and accelerated the advancement of large data analysis. HPCs are well-suited for big data analysis as they divide computational tasks among processors allowing tasks to be run in parallel both distributing the amount of computational power needed per processor and speeding up the process as these tasks are completed simultaneously. These various processors are referred to as compute nodes and an HPC can have hundreds of nodes [19]. Users can remotely log into an HPC and submit jobs to run on the various nodes for large computations. Furthermore, HPCs can have large amounts of data storage – another crucial aspect when potentially working with terabytes of sequences.

2.4 Singularity Containerization

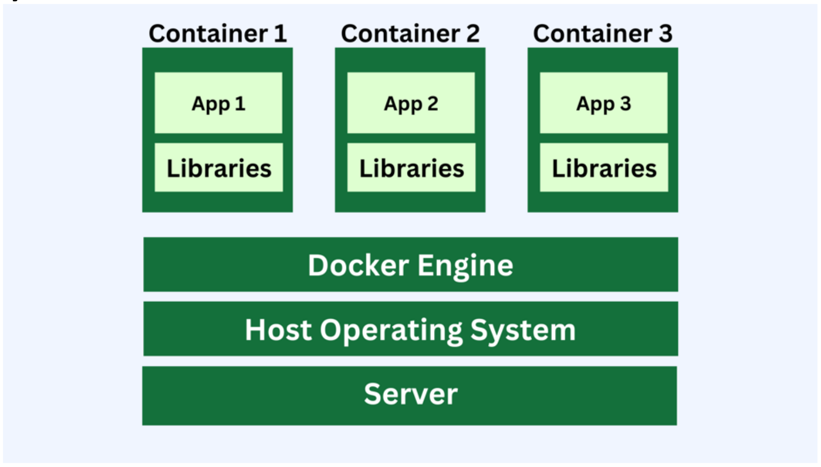


Figure 5. Architecture of Docker, a containerization platform similar to Singularity, highlighting each app within its own sub-system [20].

In bioinformatics, reproducibility has become a major focus. As technology and software have advanced, and data input files have grown in size and complexity, it is important to be able to achieve the same result multiple times. The same software with the same inputs should produce the same output so that if needed, other scientists can run the same program and get the same results and thus conclusions.

As these processes have become more complex this can become difficult. Take for example a bioinformatic analyst running program X on their computer on a single file. Now at first, it seems fairly simple for someone to reproduce this as long as they have program X downloaded and have access to the file; however, one must then consider program X's versions. What if the bioinformatic analyst did their computation with an older version of the software? Even if program X's versions are the same, the tools that are used within program X (i.e. program X's dependencies) might be different versions. What if program X cannot run on certain systems or different systems influence the way program X analyzes the file leading to slightly different results depending on the computer it is run on? This is where containerization arises.

To replicate computational results by hand becomes increasingly tedious as the many levels of dependences must be evaluated. This is instead achieved via containerization. Containers are small files containing a base operating system, various softwares, their sub-applications and version requirements [21]. In this way, software can become more portable and reproducible across machines. In this way software can be "written once and run anywhere" [22]. Container platforms, such as Singularity, use 'images', a file of software versions and dependencies, to enable reproducibility. Singularity was developed for the high-performance computing clusters (HPC) where PEGASUS must be run [23].

Nextflow, the pipeline language PEGASUS was written in, allows for each process to be run in a separate container. This allows each step of PEGASUS to be completely containerized and reproducible. This allows for easier implementation for end-users, removing the burden of having to manually download many applications while also creating a reproducible workflow. PEGASUS uses Galaxy Project containers and upon the start of each process pulls the needed container from their public container website [36]. This allows the complete run of PEGASUS with no further downloads other than PEGASUS itself.

2.5 Nextflow

Nextflow is a specialized software language tailored to efficiently execute pipelines and bioinformatic workflows. Nextflow decomposes workflows into discrete subunits termed 'processes,' with each process encapsulating a single step of the pipeline. In a typical workflow, each process contains a different software implementation. The processes are then linked together funneling the output from one step to serve as the input for the subsequent one, thereby establishing an automated workflow. Processes within Nextflow execute in parallel for enhanced efficiency and quicker runtimes. Nextflow checkpoints completed processes allowing for quick bug fixes or modification in individual processes without having to restart the entire workflow from the beginning in the event of an error. Instead, the workflow can resume from the point where it was last interrupted. This checkpointing mechanism enhances efficiency and minimizes downtime, ensuring smoother workflow management and quicker iterations during development. This step-by-step approach and checkpoint mechanism allows for the swift development of workflows comprised of diverse software tools and flexible integration of new components [49].

When executing a Nextflow pipeline, a working directory is automatically generated. Within this directory, each process is run within a unique subdirectory, containing all input, intermediate, and output files generated during each process's execution. This has many benefits for bioinformatic workflows, relieving the front-end user from the burden of managing intermediate files. In addition, Nextflow allows for relevant files to be published in a results directory outside of the work directory allowing the user to easily discern and retain only relevant output. Post-execution of a pipeline, users can effortlessly clean up unwanted files by deleting the work directory, eliminating any extraneous intermediate files and data clutter without deleting the relevant output located in the results directory [49].

Working with large files, various software applications, and numerous interconnected components it has become increasingly important to ensure reproducible results. When developing computational tools for the analysis of potentially hundreds of gigabytes of data, users should not be able to start with the same inputs and get different outcomes. Nextflow addresses this problem by allowing each process to be containerized. A container is a snapshot of software versions and dependencies that make a separate environment for each process to be run in. This ensures that regardless of the software locally installed on a user's machine a containerized Nextflow pipeline will execute the same every time allowing for reproducible results [49].

2.6 Developing User Friendly Interface

PEGASUS was designed to run through a Bash script that invokes the Nextflow pipeline. This implementation was chosen to streamline the use of PEGASUS and enhance user experience. In software development, especially when the goal is to enable others' research, a user-friendly and accessible platform is critical. The addition of the bash script was initially developed to seamlessly integrate an R script needed for filtering results from Centrifuge [29][39][40]. Nextflow requires the path to the file but asking users to specify paths to files in the PEGASUS package manually is inconvenient. The bash script integrated the pipeline and R script (along with a Nextflow configuration file) without requiring any manual input from the user. This approach makes the software more accessible, ultimately reaching and benefiting a broader audience.

Furthermore, the usage of a bash script allows for other additions such as useful crash errors and documentation. PEGASUS requires eight user-specified parameters and through this script, any missing inputs will be identified and warned. On top of this, a documentation page can be seen when PEGASUS is run with no parameters specified. The script also verifies software dependencies such as Nextflow and Singularity installations before execution, which are essential for running PEGASUS. If either dependency is not installed the program exits and indicates which software needs to be downloaded.

3 Building a Genome

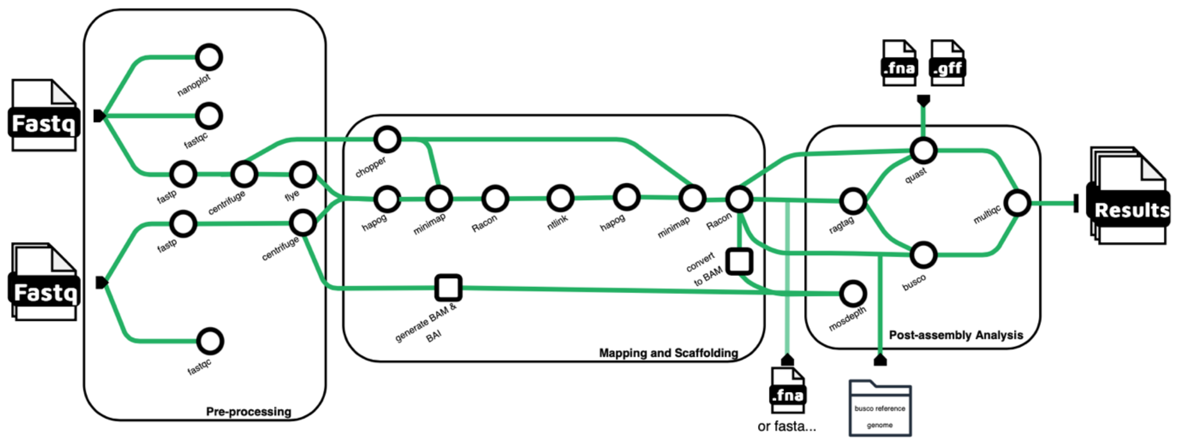


Figure 6. PEGASUS pipeline with containerized Nextflow processes

PEGASUS is an automated workflow that contains three main phases—pre-processing analysis, a mapping and scaffolding phase, and post assembly analysis. In preprocessing, raw reads are analyzed and cleaned. Once all reads are prepped the second phase begins with a rough genome build followed by alternating rounds of short and long read polishing with a scaffolding step. This results in a finished genome which is finally run through three quality assessment tools.

3.1 Preprocessing Analysis

Genome assembly requires a vast number of sequenced reads to ensure that all regions of the genome are present in assembly. PEGASUS runs all raw reads through Fastqc for an overview of read statistics and quality [30]. Long reads are also passed through Nanoplot for further analysis [37]. Due to sequencing technologies, the direct raw output of sequencing machines must be trimmed and "cleaned" before a genome draft can be put together. Initial raw reads are passed into Fastp for adapter removal – Both long and short read sequences require adapter removal. Short-read adapter removal is done via per-read overlap analysis and thus does not require a user-specified adapter sequence. The long reads use Fastp's single-end adapter trimming feature which analyzes the tails of reads to determine adapters [27].

The adapter-less or "trimmed" reads are then passed to Centrifuge. Centrifuge is a metagenomic classification software that can identify human, viral, and prokaryotic sequences within reads. Centrifuge can identify extraneous contaminants in input samples [29]. Flagged reads are then removed from samples via Seqtk and an R script [38][39][40]. PEGASUS eliminates all human, prokaryotic, or viral reads that may have made it into the sequencer with the sample.

3.2 Genome Assembly

Once the raw reads have gone through pre-processing a draft genome is built. Flye is a long-read scaffolding and polishing tool and is implemented to build an initial draft with long reads only. Flye also outputs read-analysis – prominently long-read coverage [31].

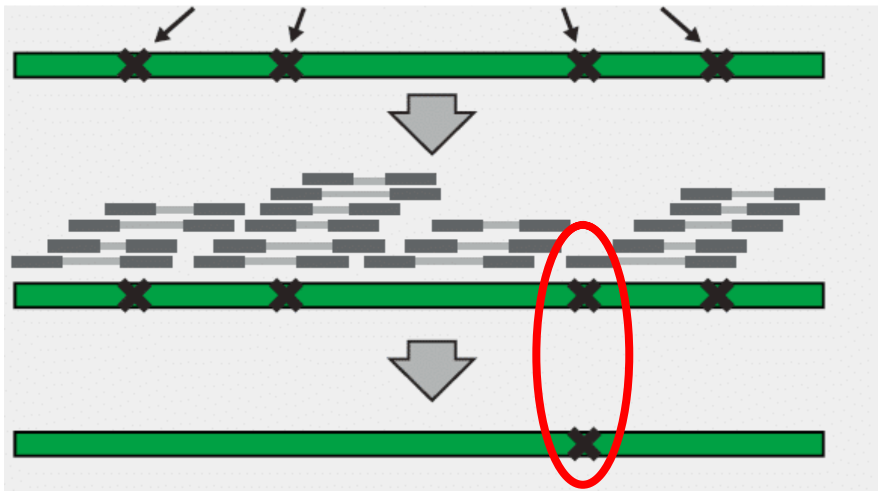


Figure 7. Genome polishing to correct errors (shown by X) of intermediate genome by aligning reads and comparing bases. Regions of high read depth are corrected but is unsuccessful in low depth region (as seen in red) [24].

As the Flye-assembled genome is built entirely from long-reads, the genome is then polished with the short-reads using Hapo-g [31][32]. Short-reads typically have higher quality scores and thus are used to validate the long-read-only draft from Flye [25]. Polishing is the process in which errors are corrected in the draft genome by aligning and comparing genome sequences to raw reads.

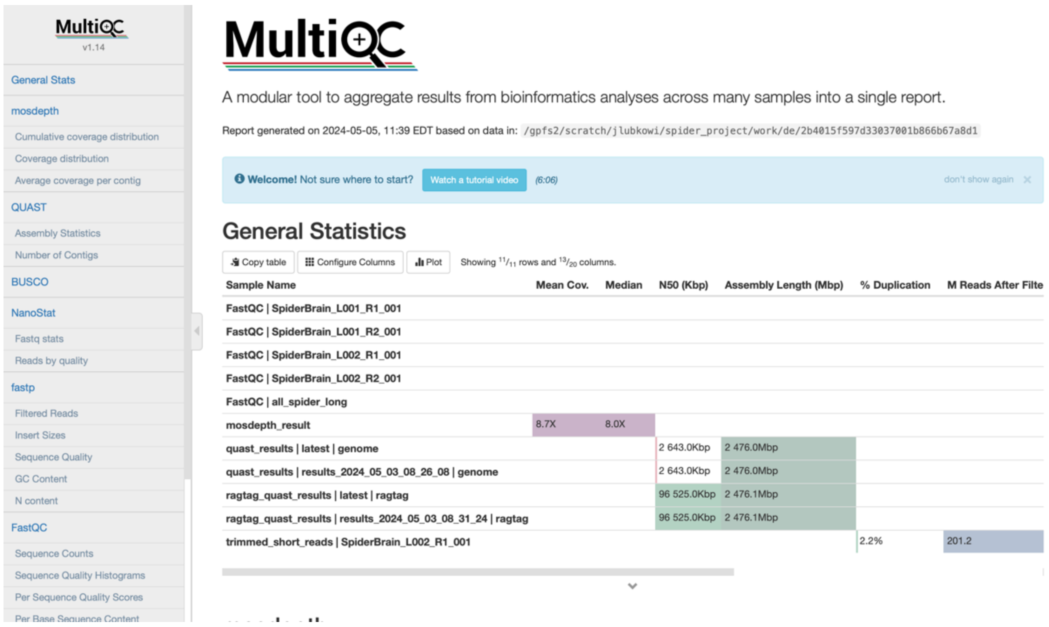
To further strengthen the drafted genome another round of polishing is done with the long reads. While Hapo-g does have a long-read polishing feature it is only recommended to be used with PacBio HIFI reads [32]. As long-read sequences have higher error rates compared to short-reads the long reads with quality scores (q-scores) less than 20 are filtered out before use in polishing. This is done with Chopper, a software for filtering reads [37]. The reads are then mapped onto the genome draft with Minimap2 [41]. Once each sequence is mapped onto the genome Racon is used to polish with all quality long reads [42].

The genome has now been polished with all short reads and all quality long reads. The raw long reads are mapped and scaffolded onto the draft via Ntlink [33]. In doing so, longer congruent contigs are built as raw reads that span two short contigs are used to "bridge the gap" to create one continuous sequence or contig. As scaffolding with raw reads is an opportunity to introduce errors, the genome is then repolished with first short-reads and then long-reads in the same manner as the first polishing phases (Hapo-g and Racon).

Upon completion of the second round of polishing a 'built from scratch' (ie. read-only) genome is complete. PEGASUS offers an additional round of scaffolding with a reference genome via Ragtag for increased genome quality [34]. This offers the potential to increase the quality of the genome assembly without further sequencing samples or increased input reads. Reference genomes should be as closely related to the organism being assembled as possible and could lead to improved assembly. This has the potential to decrease sequencing costs when building many genomes of one species. This was implemented in the development of PEGASUS when building 8 catfish genomes. One individual was heavily sequenced (~165GB of total reads) and assembled. This reference genome was then used as a scaffold in Ragtag when building the other genomes (which had about 60GB of sequences per sample). This extra scaffolding step with a reference genome represents a chance to construct a quality viable genome without excess sequencing though it will have lower read coverage.

3.3 Post assembly Analysis

PEGASUS results in the construction of two genomes; the read-only build and the reference-genome-scaffolded genome. Depending on the relatedness and quality of the reference genome, the ragtag genome can have increased quality metrics than the read-only build [34]. Both genomes are then assessed via Busco, Quast, and Mosdepth. Busco assesses genome quality using single copy orthologs [13][14]. Quast collects metrics including contig lengths and N50 score, in addition to comparison to a reference genome both by sequence and GC content [15]. The short reads are converted to a BAM file via BWA and Samtools and then used with the read-only genome to assess short read depth coverage with Mosdepth [45][52].



A modular tool to aggregate results from bioinformatics analyses across many samples into a single report.

Report generated on 2024-05-05, 11:39 EDT based on data in: `/gpfs2/scratch/jlubkowi/spider_project/work/de/2b4015f597d33037001b866b67a8d1`

General Statistics

Showing $^{11}/_{11}$ rows and $^{13}/_{20}$ columns.

Sample Name	Mean Cov.	Median	N50 (Kbp)	Assembly Length (Mbp)	% Duplication	M Reads After Filte
FastQC | SpiderBrain_L001_R1_001						
FastQC | SpiderBrain_L001_R2_001						
FastQC | SpiderBrain_L002_R1_001						
FastQC | SpiderBrain_L002_R2_001						
FastQC | all_spider_long						
mosdepth_result	8.7X	8.0X				
quast_results | latest | genome			2 643.0Kbp	2 476.0Mbp		
quast_results | results_2024_05_03_08_26_08 | genome			2 643.0Kbp	2 476.0Mbp		
ragtag_quast_results | latest | ragtag			96 525.0Kbp	2 476.1Mbp		
ragtag_quast_results | results_2024_05_03_08_31_24 | ragtag			96 525.0Kbp	2 476.1Mbp		
trimmed_short_reads | SpiderBrain_L002_R1_001					2.2%	201.2

Figure 8. MultiQC html report from PEGASUS execution of Spider Assembly

MultiQC compiles an HTML report summarizing all output statistics from Fastp, Nanoplot, FastQC, Busco, and Quast [17]. This report provides a comprehensive overview of the entire run. Additionally, all relevant files from each process are saved in a results directory within the directory where PEGASUS was executed.

3.4 Software Versions

Through the Nextflow structure, each process is run in a separate container. The software versions within each container are detailed below. All containers are sourced from the Galaxy Project [36].

Nanoplot:1.41.0	Long read analysis tool
FastQC:0.11.9	Long and short read analysis tool
Fastp:0.23.4	Adapter removal tool for all reads
Centrifuge:1.0.4	Identify contaminate reads in sample
R-Tidyverse:3A1.2.1	Used to select non-contaminated reads w/ Centrifuge
Seqtk:1.3	Used to edit genomic file formatting
Flye:2.9	Long read genome assemble tool
Hapo-g:1.3.7	Short read genome polishing tool
Chopper:0.7.0	Quality read filter tool
Minimap2:2.26	Long read alignment tool
Racon:1.5.0	Long read genome polishing tool
Ntlink:1.3.9	Long read scaffolding tool
Ragtag:2.1.0	Reference genome scaffolding tool
Quast:5.2.0	Genome assembly analysis tool
Busco:5.5.0	Genome assembly analysis tool
Bwa-mem2:2.2.1	Used to convert file types
Mosdepth:0.3.6	Short read depth tool
Multiqc:1.14	Used to generate final html report

3.5 Software Selection and Optimization

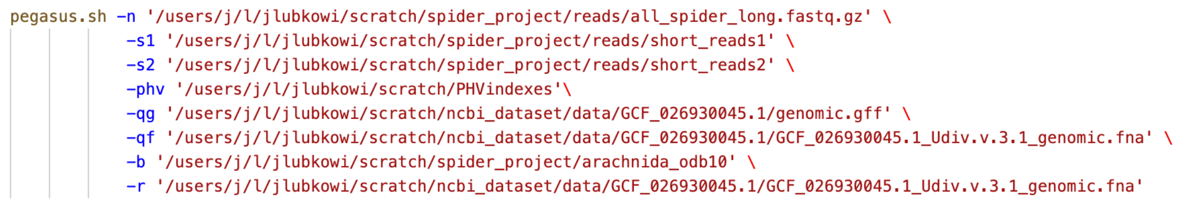
During the development of PEGASUS, we experimented with various software options for the processes. The two most significant changes were the addition of long-read polishing and the choice of software, as well as switching the adapter trimming software. These changes were driven by a need to decrease computational resources such as memory use and a desire to speed up the pipeline process. Initially, we used Trimmomatic for trimming short reads and Porechop_abi for long reads. While this software performed well for smaller runs, processing larger raw long-read files with Porechop_abi significantly slowed down the process, required substantial memory, and often exceeded memory constraints to the point of crashing. To improve the adapter removal step, we replaced both Trimmomatic and Porechop_abi with Fastp. Fastp is a "fast all-in-one preprocessing [tool]… with multithreading supported to afford high performance" [27]. The addition of Fastp not only decreased the time it took to remove adapters but also used less memory making the pipeline more accessible and efficient and decreasing the likelihood of a crash.

Another major addition was polishing our intermediate genome assemblies with the long reads. In early development, we used the long-reads for the initial assembly and later scaffolding with only short-read polishing steps between. This provided adequate genome assembly but for further error correction, we added the long-read polishes to each place we also short-read polished. As long reads are more error-prone, we used only long reads with quality scores above 20. Hapo-g, the short-read polishing software, also offers a long-read polish option but with the amount of long-read data in genome assembly, much like with adapter processing, it has some memory issues [32].

The crashing of Hapo-g due to exceeding memory limits prompted us to try other long-read polishing software including Pilon, Racon, and Medaka. Both Pilon and Racon use error correction analysis to polish the input genome while Medaka implements a neural network to correct errors in the genome based on inputted raw reads. All three were run on the same intermediate genome which consisted of a Flye assembled and Hapo-g short read polished genome. To compare the results of the three we used runtime, Quast, and Busco metrics to check the accuracy. All three software results in very similar genomes with almost no difference in Busco score, N50, polished genome size, etc. The largest difference between the three was in Racon runtime and the number of contigs remaining after polishing. Racon was significantly faster than any other long read polisher taking ∼1 hour to polish a 2.7GB genome with ∼31GB of long reads compared to Medaka's ∼11 hours and Pilon's 24+ hours. Despite the significant difference in runtime, Racon maintained the same quality output as the others (and even had the highest Busco rating by 0.1). Racon also eliminated shorter contigs from the genome (typically ones under 1000bp and some under 5000bp). All three software require large amounts of memory given the size of the input files. With that in mind, Racon's peak memory usage (peak_vmem) was 113.6 GB with a total of 108.2GB data read during the process (rchar) and 2.3GB written to the disk (wchar) (ie just the genome). Medaka, on the other hand, had a peak memory usage (peak_vmem) of 154.9GB with 2.8TB of data read (rchar) and 597.6 GB written to the disk (wchar). Based on these findings in both memory usage and runtime we implemented Racon in our pipeline for all long-read polishing [42][43][44].

4 Usage and Implementation

4.1 User input



```
pegasus.sh -n '/users/j/l/jlubkowi/scratch/spider_project/reads/all_spider_long.fastq.gz' \
            -s1 '/users/j/l/jlubkowi/scratch/spider_project/reads/short_reads1' \
            -s2 '/users/j/l/jlubkowi/scratch/spider_project/reads/short_reads2' \
            -phv '/users/j/l/jlubkowi/scratch/PHVindexes'\
            -qg '/users/j/l/jlubkowi/scratch/ncbi_dataset/data/GCF_026930045.1/genomic.gff' \
            -qf '/users/j/l/jlubkowi/scratch/ncbi_dataset/data/GCF_026930045.1/GCF_026930045.1_Udiv.v.3.1_genomic.fna' \
            -b '/users/j/l/jlubkowi/scratch/spider_project/arachnida_odb10' \
            -r '/users/j/l/jlubkowi/scratch/ncbi_dataset/data/GCF_026930045.1/GCF_026930045.1_Udiv.v.3.1_genomic.fna'
```

Figure 9. Example of PEGASUS pipeline run with parameters.

PEGASUS requires eight parameters for a complete run. The path to the long-read raw file should be specified using the -n parameter. The short reads are provided as two batches of paired-end reads using the -s1 and -s2 parameters. These parameters should point to directories containing paired-end read files named according to the typical short-read naming convention (*_R1_001.fastq.gz and *_R2_001.fastq.gz). The -phv parameter specifies the path to the Centrifuge index files used to classify extraneous human, prokaryotic, or viral sequences in the sample. Centrifuge indexes can be downloaded from the Centrifuge GitHub page. For post-assembly quality analysis, a reference genome is required and specified with paths to a GFF file and an FNA file using the -qg and -qf parameters. These files can be acquired from NCBI databases. BUSCO, a quality assembly metric, requires a reference input to compare single-copy orthologs, specified via the -b parameter. The last parameter, -r, is a reference genome used for further scaffolding, which should be provided in FASTA, or FASTQ format [53].

4.2 Dependencies and Requirements

PEGAUS is an open-source pipeline, meaning that anyone can use it by downloading it from the internet. PEGASUS can be found on GitHub under an MIT license and pulled for usage. PEGASUS was designed with user implementation in mind to enable full genome assembly with maximum automation and organization of intermediate data processing steps. It is our hope that PEGASUS is used widely to advance research beyond the scope in which we have implemented it (i.e., assembling a Catfish and Spider genome).

PEGASUS has only a handful of requirements and dependencies to run. PEGASUS requires both Nextflow and Singularity to be installed and given the size of the data computational tasks must be run on an HPC or equivalent with enough power. The nodes, CPU, and memory required to run PEGASUS are highly influenced by the number of raw reads used to assemble the genome and the size of the genome being assembled. In development, we used 1 node with 40 CPUs per task and 256GB of memory to assemble a genome of ~2.3GB from ~122GB of long-reads and ~50GB of short-reads. As the memory is decreased, many of these tools will have trouble functioning as they store reads in memory while executing. To avoid this, it is recommended to round up when allocating computational resources for an execution. Given the complexity of the task and size of files, PEGASUS can take multiple days to run. As each process is completed it is cached and can be later resumed if computational requirements should falter.

4.3 Result Output

```
Results
|   -- Busco
|   |   -- busco_genome
|   |   |   -- Contains Busco results of hapog-built genome
|   |   -- busco_ragtag
|   |   |   -- Contains Busco results of ragtag-scaffolded genome
|   -- Fastp_long
|   |   |   -- Adapterless long reads
|   -- FastQC
|   |   -- Contains FastQC zip and html files
|   -- Flye
|   |   -- flye
|   |   |   -- Contains assembly.fasta, an intermediate genome
|   |   -- contig_stats.txt, contians long read coverage
|   -- Hapog1
|   |   |  -- Contains hapog.fasta an intermediate genome
|   -- Hapog2
|   |   -- Contains hapog.fasta an intermediate genome
|   -- long_centrifuge
|   |   -- Contains cleaned longreads in fastq.gz
|   -- Mosdepth
|   |   -- Contains mosdepth result files
|   -- MultiQC
|   |   -- MultiQC report
|   -- Nanoplot
|   |   -- Contains Nanoplot png and html files
|   -- quast_results
|   |   -- results from read-only genome Quast
|   -- NTLink
|   |   -- Contains hapog_result.fasta.k32.w250.z1000.ntLink.5rounds.fa, an intermediate genome
|   -- RagTag
|   |   -- Contains all ragtag results
|   -- Racon2
|   |   -- final genome from read-assembly
|   -- ragtag_quast_results
|   |   |-- results from Ragtag genome Quast
|   -- short_centrifuge
|   |   -- short_centrifuge_results
|   |   |   -- Contains cleaned shortreads in fastq.gz
|   |   -- short read centrifuge stats
|   -- trimmed_short_reads
|   |   -- Fastp adapterless short reads
```

Figure 10. Result Directory Structure of PEGASUS

PEGASUS outputs a ‘Results’ directory via Nextflow of all relevant files. If other intermediate files are desired, they can be found in the ‘work’ directory where Nextflow executed. Each process has its own directory inside of ‘Results’ containing the all output. The final genomes (one read-only build and one ragtag scaffolded build) can be found in ‘Results/Racon2’ and ‘Results/Ragtag’. A MultiQC generated report containing assembly metrics can be found in

‘Results/MultiQC ‘. Furthermore, long read depth coverage can be seen in the Results/Flye directory while short read coverage can be found in MultiQC or Mosdepth directories.

6 Conclusion

This paper introduces PEGASUS, an automated and streamlined hybrid genome assembly tool. We delved into the biological background needed to effectively work with DNA sequences including sequencing techniques, read depth metrics, and polishing/scaffolding techniques. We then examined how these factors played a role in the context of our project, prompting a delve in to the computational elements that enable reproducible, parallelized, and efficient analysis. With this foundation, we outlined the steps of our genome assembly process and the rationale behind selecting our design approach of fast and efficient tools. Finally, we explored how to integrate PEGASUS into various workflows, including managing outputs and results to support broader adoption.

7 References

1. Supple, M.A., Shapiro, B. Conservation of biodiversity in the genomics era. Genome Biol 19, 131 (2018). https://doi.org/10.1186/s13059-018-1520-3
2. Genomics and our future food security. Nat Genet 51, 197 (2019). https://doi.org/10.1038/s41588-019-0352-8
3. Pattan V, Kashyap R, Bansal V, Candula N, Koritala T, Surani S. Genomics in medicine: A new era in medicine. World J Methodol. 2021 Sep 20;11(5):231-242. doi: 10.5662/wjm.v11.i5.231. PMID: 34631481; PMCID: PMC8472545.
4. Genomics. Office of Science. Energy.gov. https://www.energy.gov/science/genomics#:~:text=Genomic%20science%20is%20now%20an,used%20to%20improve%20industrial%20bioprocesses.
5. Marks, R.A., Hotaling, S., Frandsen, P.B. et al. Representation and participation across 20 years of plant genome sequencing. Nat. Plants 7, 1571–1578 (2021). https://doi.org/10.1038/s41477-021-01031-8
6. NHS. Long-read sequencing. NHS choices. https://www.genomicseducation.hee.nhs.uk/genotes/knowledge-hub/long-read-sequencing/
7. Structure of DNA. Labster. https://theory.labster.com/structure-dna/
8. Heather JM, Chain B. The sequence of sequencers: The history of sequencing DNA. Genomics. 2016 Jan;107(1):1-8. doi: 10.1016/j.ygeno.2015.11.003. Epub 2015 Nov 10. PMID: 26554401; PMCID: PMC4727787.
9. Sequencing quality scores. Illumina. https://www.illumina.com/science/technology/next-generation-sequencing/plan-experiments/quality-scores.html
10. Scholz M. Coverage depth. Metagenomics. 2022 May. https://www.metagenomics.wiki/pdf/qc/coverage-read-depth
11. Genomics Education Programme. Read depth . NHS England. https://www.genomicseducation.hee.nhs.uk/glossary/read-depth/#:~:text=The%20number%20of%20times%20a,known%20as%20’base%20calling
12. Ou, S., Liu, J., Chougule, K.M. et al. Effect of sequence depth and length in long-read assembly of the maize inbred NC358. Nat Commun 11, 2288 (2020). https://doi.org/10.1038/s41467-020-16037-7
13. Manni, M., Berkeley, M., Seppey, M., Simão, F., Zdobnov, E. BUSCO Update: Novel and Streamlined Workflows along with Broader and Deeper Phylogenetic Coverage for Scoring of Eukaryotic, Prokaryotic, and Viral Genomes. Molecular Biology and Evolution, Volume 38, Issue 10, October 2021, Pages 4647–4654
14. Manni, M., Berkeley, M. R., Seppey, M., & Zdobnov, E. M. (2021). BUSCO: Assessing genomic data quality and beyond. Current Protocols, 1, e323. doi: 10.1002/cpz1.323
15. Gurevich, A., Saveliev, V., Vyahhi, N., Tesler, G. QUAST: quality assessment tool for genome assemblies, Bioinformatics, Volume 29, Issue 8, April 2013, Pages 1072–1075, https://doi.org/10.1093/bioinformatics/btt086

16. CD Genomics. An Overview of Genome Assembly. The Genomics Services Company. https://www.cd-genomics.com/an-overview-of-genome-assembly.html#:~:text=Once%20the%20reference%20genome%20is,Table%201.
17. Ewels, P., Magnusson, M., Lundin, S., Käller, M. MultiQC: summarize analysis results for multiple tools and samples in a single report, Bioinformatics, Volume 32, Issue 19, October 2016, Pages 3047–3048, https://doi.org/10.1093/bioinformatics/btw354
18. Hbctraining. Hbctraining/intro-to-shell-flipped. GitHub. https://github.com/hbctraining/Intro-to-shell-flipped
19. IBM. What is high-performance computing (HPC)?. IBM. https://www.ibm.com/topics/hpc
20. Potdar, A. Containerization: A beginner's guide to its impact on software development. Bito. https://bito.ai/blog/containerization-a-beginners-guide-to-its-impact-on-software-development/
21. Definition files. Definition Files - Singularity container 3.5 documentation. https://docs.sylabs.io/guides/3.5/user-guide/definition_files.html
22. IBM. What is containerization?. IBM. https://www.ibm.com/topics/containerization
23. Introduction files. Introduction Files - Singularity container 3.5 documentation. https://docs.sylabs.io/guides/3.5/user-guide/definition_files.html
24. Casey, R. How to generate a complete genome with hybrid assembly. The Sequencing Center. https://thesequencingcenter.com/knowledge-base/complete-genome-assembly/
25. Amarasinghe, S.L., Su, S., Dong, X. et al. Opportunities and challenges in long-read sequencing data analysis. Genome Biol **21**, 30 (2020). https://doi.org/10.1186/s13059-020-1935-5
26. Wikimedia Foundation. (2024, April 30). Hybrid genome assembly. Wikipedia. https://en.wikipedia.org/wiki/Hybrid_genome_assembly
27. Shifu Chen. 2023. Ultrafast one-pass FASTQ data preprocessing, quality control, and deduplication using fastp. iMeta 2: e107. https://doi.org/10.1002/imt2.107
28. Nussinov R, Tsai CJ, Jang H. Protein ensembles link genotype to phenotype. PLoS Comput Biol. 2019 Jun 20;15(6):e1006648. doi: 10.1371/journal.pcbi.1006648. PMID: 31220071; PMCID: PMC6586255.
29. Kim D, Song L, Breitwieser FP, and Salzberg SL. Centrifuge: rapid and sensitive classification of metagenomic sequences. Genome Research 2016 http://ccb.jhu.edu/software/centrifuge/
30. Andrews S. FastQC: a quality control tool for high throughput sequence data.2010. https://www.bioinformatics.babraham.ac.uk/projects/fastqc/
31. Lin, Y., Yuan, J., Kolmogorov, M., Shen, M., Chaisson, M., Pevzner, P. "Assembly of Long Error-Prone Reads Using de Bruijn Graphs", PNAS, 2016 doi:10.1073/pnas.1604560113
32. Aury, J., Istace, B. Hapo-G, haplotype-aware polishing of genome assemblies with accurate reads, NAR Genomics and Bioinformatics, Volume 3, Issue 2, June 2021, lqab034, https://doi.org/10.1093/nargab/lqab034
33. Coombe L, Li JX, Lo T, Wong J, Nikolic V, Warren RL and Birol I. LongStitch: High-quality genome assembly correction and scaffolding using long reads. bioRxiv. 2021;2021.06.17.448848. doi: https://doi.org/10.1101/2021.06.17.448848

34. Alonge, Michael, et al. "Automated assembly scaffolding elevates a new tomato system for high-throughput genome editing." Genome Biology (2022). https://doi.org/10.1186/s13059-022-02823-7
35. Narzisi, B., Mishra. Comparing De Novo Genome Assembly: The Long and Short of It. Plos One, 2011, https://doi.org/10.1371/journal.pone.0019175
36. The Galaxy Community. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2022 update, Nucleic Acids Research, Volume 50, Issue W1, 5 July 2022, Pages W345–W351, doi:10.1093/nar/gkac247
37. De Coster, W. Rademakers, R. NanoPack2: population-scale evaluation of long-read sequencing data, *Bioinformatics*, Volume 39, Issue 5, May 2023, btad311, https://doi.org/10.1093/bioinformatics/btad311
38. H. Li, Seqtk: a fast and lightweight tool for processing FASTA or FASTQ sequences, 2013. https://github.com/lh3/seqtk
39. R Core Team (2021). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/.
40. Wickham H, Averick M, Bryan J, Chang W, McGowan LD, François R, Grolemund G, Hayes A, Henry L, Hester J, Kuhn M, Pedersen TL, Miller E, Bache SM, Müller K, Ooms J, Robinson D, Seidel DP, Spinu V, Takahashi K, Vaughan D, Wilke C, Woo K, Yutani H (2019). "Welcome to the tidyverse." *Journal of Open Source Software*, **4**(43), 1686. doi:10.21105/joss.01686.
41. Li, H. (2018). Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics, 34:3094-3100. doi:10.1093/bioinformatics/bty191
42. R. Vaser, I. Sović, N. Nagarajan, M. Šikić, Fast and accurate de novo genome assembly from long uncorrected reads. Genome Res. 27, 737–746 (2017).
43. Walker, B., Abeel, T., Shea, T., Priest, M., Abouelliel A., Sakthikumar S., Cuomo C., Zeng, Q., Wortman, J., Young S., Earl, A. (2014) Pilon: An Integrated Tool for Comprehensive Microbial Variant Detection and Genome Assembly Improvement. PLoS ONE 9(11): e112963. doi:10.1371/journal.pone.0112963
44. Medaka, O. N. T. (2018). sequence correction provided by ONT Research. Github. https://github. com/nanoporetech/medaka.
45. Pedersen, B., Quinlan, A. Mosdepth: quick coverage calculation for genomes and exomes, Bioinformatics, Volume 34, Issue 5, March 2018, Pages 867–868, https://doi.org/10.1093/bioinformatics/btx699
46. Jiang, M., Bu, C., Zeng, J. et al. Applications and challenges of high performance computing in genomics. CCF Trans. HPC 3, 344–352 (2021). https://doi.org/10.1007/s42514-021-00081-w
47. Xavier, B.B., Sabirova, J., Pieter, M. et al. Employing whole genome mapping for optimal de novo assembly of bacterial genomes. BMC Res Notes 7, 484 (2014). https://doi.org/10.1186/1756-0500-7-484
48. Chaney, L., Sharp, A., Evans, C., Udall, J. Genome Mapping in Plant Comparative Genomics. 50 Trends in Plant Science, Volume 21, Issue 9, September 2016, Pages 770-780. https://doi.org/10.1016/j.tplants.2016.05.004

49. Di Tommaso, P., Chatzou, M., Floden, E. W., Barja, P. P., Palumbo, E., & Notredame, C. Nextflow enables reproducible computational workflows. Nature Biotechnology, 35(4), 2017, 316–319. doi:10.1038/nbt.3820
50. Rice, E., Green, R. New Approaches for Genome Assembly and Scaffolding. ANNUAL REVIEW OF ANIMAL BIOSCIENCES Volume 7:17, 2019. https://doi.org/10.1146/annurev-animal-020518-115344
51. Junwei Luo, Yawei Wei, Mengna Lyu, Zhengjiang Wu, Xiaoyan Liu, Huimin Luo, Chaokun Yan, A comprehensive review of scaffolding methods in genome assembly, Briefings in Bioinformatics, Volume 22, Issue 5, September 2021, bbab033, https://doi.org/10.1093/bib/bbab033
52. Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., and 1000 Genome Project Data Processing Subgroup. 2009. The Sequence Alignment/Map format and SAMtools. Bioinformatics 25(16): 2078-2079. doi://10.1093/bioinformatics/btp352
53. Cock, P., Fields, C., Goto, N., Heuer, M., Rice, P., The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. Nucleic Acids Res. 2010 Apr;38(6):1767-71. doi: 10.1093/nar/gkp1137. Epub 2009 Dec 16. PMID: 20015970; PMCID: PMC2847217.

8 PEGASUS Code

8.1 Bash Calling Script

```
#!/bin/bash

nanoPath=""
shortPath1=""
shortPath2=""
quastPathGFF=""
quastPathFNA=""
buscoPath=""
ref1=""
rpath=""
phv=""

usage() {
    echo "Usage: $0 [-n <value>] [-s1 <value>] [-s2 <value>] [-
qg <value>] [-qf <value>] [-b <value>] [-r <value>] [-phv
<value>]"
    echo "Options:"
    echo "  -n <value>: Path to Nanopore Long Reads"
    echo "  -s1 <value>: Path to Short Reads 1"
    echo "  -s2 <value>: Path to Short Reads 2"
    echo "  -qg <value>: Path to Quast GFF file"
    echo "  -qf <value>: Path to Quast FNA file"
    echo "  -b <value>: Path to BUSCO Reference"
    echo "  -r <value>: Path to reference genome for Ragtag"
    echo "  -phv <value>: Path to PHV files for Centrifuge"
    exit 1
}

check_command() {
    if command -v "$1" &>/dev/null; then
        echo "$1 is installed."
    else
        echo "$1 is not installed."
        echo "Please download and install $1 to proceed."
        exit 1
    fi
}

check_singularity() {
    echo "Checking for Singularity..."
    check_command "singularity"
}

check_nextflow() {
    echo "Checking for Nextflow..."
    check_command "nextflow"
}

check_singularity
check_nextflow
while [[ $# -gt 0 ]]; do
    key="$1"
    case $key in
        -n)
            nanoPath="$2"
            shift
            shift
            ;;
        -s1)
            shortPath1="$2"
            shift
            shift
            ;;
        -s2)
            shortPath2="$2"
            shift
            shift
            ;;
        -qg)
            quastPathGFF="$2"
            shift
            shift
            ;;
        -qf)
            quastPathFNA="$2"
            shift
            shift
            ;;
        -b)
            buscoPath="$2"
            shift
            shift
            ;;
        -r)
            ref1="$2"
            shift
            shift
            ;;
         -phv)
            phv="$2"
            shift
            shift
            ;;
        *)
```

```
            usage
            ;;
    esac
done

script_dir=$(dirname "$0")

pegasus_nextflow=$(echo "$script_dir/bin/pegasus.nf")
rpath=$(echo "$script_dir/bin/run_centrifuge_clean.R")

if [[ -n $nanoPath && -n $shortPath1 && -n $shortPath2 && -n
$quastPathGFF && -n $quastPathFNA && -n $buscoPath && -n $ref1
&& -n $phv ]]; then
    echo "Configuration 1: Long- and Short-Read Assembly"
    parameters=$(echo "-resume
    --nanoPath $nanoPath
    --shortPath1 $shortPath1
    --shortPath2 $shortPath2
    --quastPathGFF $quastPathGFF
    --quastPathFNA $quastPathFNA
    --buscoPath $buscoPath
    --ref1 $ref1
    --centrifugeRscript $rpath
    --phvDatabase $phv")

    execute=$(echo "nextflow run $pegasus_nextflow $parameters")
    $execute
else
    echo "Illegal configuration: Incomplete flags entered"
    usage
fi
```

8.2 Nextflow Pipeline

```
#!/usr/bin/env nextflow

// initialzie and declare input channels
nextflow.enable.dsl=2
params.publish_dir = './Results'

params.nanoPath = ""
params.shortPath1 = ""
params.shortPath2 = ""
params.buscoPath = ""
params.quastPathFNA = ""
params.quastPathGFF = ""
params.phvDatabase = ""
params.ref1 = ""
params.centrifugeRscript =
"/users/j/l/jlubkowi/scratch/PEGASUS/run_centrifuge_clean.R"
centrifugeRscript = Channel.fromPath( params.centrifugeRscript,
checkIfExists: true)

phvDatabase = Channel.fromPath( params.phvDatabase,
checkIfExists: true)
shortreads1_ch = Channel.fromPath( params.shortPath1,
checkIfExists: true )
shortreads2_ch = Channel.fromPath( params.shortPath2,
checkIfExists: true )
longread_ch = Channel.fromPath( params.nanoPath , checkIfExists:
true)
quastpathGFF_ch = Channel.fromPath( params.quastPathGFF,
checkIfExists: true)
quastpathFNA_ch = Channel.fromPath( params.quastPathFNA,
checkIfExists: true)
buscopath_ch = Channel.fromPath( params.buscoPath,
checkIfExists: true)
ref1_ch = Channel.fromPath( params.ref1, checkIfExists: true)

// pipeline processes

process NANOPLOT {
    publishDir "${params.publish_dir}", mode: 'copy'

    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?

'https://depot.galaxyproject.org/singularity/nanoplot:1.41.0--
pyhdfd78af_0' :
        'biocontainers/nanoplot:1.41.0--pyhdfd78af_0' }"

    input:
        path nano_read
    output:
        path 'Nanoplot'
    script:
        """
        mkdir Nanoplot
        cp ${nano_read} Nanoplot
        cd Nanoplot
        NanoPlot -t 40 --fastq ${nano_read}
        """
}

process FASTQC {
    publishDir "${params.publish_dir}", mode: 'copy'

    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?

'https://depot.galaxyproject.org/singularity/fastqc:0.11.9--0' :
        'biocontainers/fastqc:0.11.9--0' }"

    input:
        path short_reads1
        path short_reads2
        path nano_reads

    output:
        path 'FastQC'
    script:
    """
        mkdir FastQC
        fastqc --noextract --nogroup -o FastQC ${short_reads1}/*
        fastqc --noextract --nogroup -o FastQC ${short_reads2}/*
        fastqc --noextract -o FastQC ${nano_reads} -t 8
    """
}

process FASTP_LONG {
     publishDir "${params.publish_dir}/Fastp_long", mode: 'copy'

    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?

'https://depot.galaxyproject.org/singularity/fastp:0.23.4--
hadf994f_3':
        'biocontainers/fastp:0.23.4--hadf994f_3' }"

    memory { 2.GB * task.attempt }
    errorStrategy { task.exitStatus in 137..140 ? 'retry' :
'terminate' }
    maxRetries 3

    input:
        path long_reads

    output:
        path 'long_reads_clean.fastq.gz'

    script:
    """
    fastp -i ${long_reads} -o long_reads_clean.fastq.gz --thread
10
    """
}

process LONG_CENTRIFUGE {
    publishDir "${params.publish_dir}/long_centrifuge", mode:
'copy'

    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?

'https://depot.galaxyproject.org/singularity/centrifuge:1.0.4_be
ta--h9a82719_6' :
        'biocontainers/centrifuge:1.0.4_beta--h9a82719_6' }"

    input:
        path longreads
        path database

    output:
        path 'long_cent_out.txt'
        path 'longreads.fastq'

    script:
    """
        cp ${longreads} \$PWD/longreads.fastq.gz
        cp ${database}/* \$PWD

        centrifuge -q -x p+h+v -U longreads.fastq.gz --threads 8
-S long_cent_out.txt --report-file long_cent_out.tsv --min-
hitlen 250

        gunzip longreads.fastq.gz
    """
}

process R_PROCESSING_LONG {
    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?
        'https://depot.galaxyproject.org/singularity/r-
tidyverse%3A1.2.1' :
        'biocontainers/r-tidyverse%3A1.2.1' }"

    input:
        path rscript
        path long_input_txt
        path long_fasta
    output:
        path 'not_contam.txt'
        path 'longreads.fastq'
    script:
    """
    cp ${rscript} \$PWD/centrifugeClean.R

    Rscript --vanilla centrifugeClean.R long_cent_out.txt
longreads
    """
}

process LONG_SEQTK {

    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?
        'https://depot.galaxyproject.org/singularity/seqtk:1.3--
h5bf99c6_3' :
        'biocontainers/seqtk:1.3--h5bf99c6_3' }"

    input:
        path sequence
        path contam
    output:
        path 'clean_longreads.fastq.gz'
    script:
    """
    seqtk subseq longreads.fastq not_contam.txt >
clean_longreads.fastq

    gzip clean_longreads.fastq
    """
}

process FLYE {

    publishDir "${params.publish_dir}/Flye", mode: 'copy'

    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?
        'https://depot.galaxyproject.org/singularity/flye:2.9--
py39h6935b12_1' :
        'biocontainers/flye:2.9--py39h6935b12_1' }"

    input:
        path nano_read

    output:
         file 'flye/assembly.fasta'
        file 'contig_stats.txt'
    script:
        """
        flye --nano-hq ${nano_read} --out-dir flye --threads 40
        cp \$PWD/flye/assembly.fasta \$PWD/assembly.fasta
        tail \$PWD/flye/flye.log > contig_stats.txt
        """
}

process FASTP_SHORT {
     publishDir "${params.publish_dir}", mode: 'copy'

    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?

'https://depot.galaxyproject.org/singularity/fastp:0.23.4--
hadf994f_3':
        'biocontainers/fastp:0.23.4--hadf994f_3' }"

    memory { 2.GB * task.attempt }
    errorStrategy { task.exitStatus in 137..140 ? 'retry' :
'terminate' }
    maxRetries 3

    input:
        path shortreads1
        path shortreads2
    output:
        path 'trimmed_short_reads'

    script:
    """
    mkdir trimmed_short_reads
    cp -r ${shortreads1} trimmed_short_reads
    cp -r ${shortreads2} trimmed_short_reads
    cd trimmed_short_reads
    fastp -i ${shortreads1}/*_R1_001.fastq.gz -I
${shortreads1}/*_R2_001.fastq.gz -o
short_reads.1_1.trim.fastq.gz -O short_reads.1_2.trim.fastq.gz -
-thread 40
    fastp -i ${shortreads2}/*_R1_001.fastq.gz -I
${shortreads2}/*_R2_001.fastq.gz -o
short_reads.2_1.trim.fastq.gz -O short_reads.2_2.trim.fastq.gz -
-thread 40
    """
}
```

```
process SHORT_CENTRIFUGE {
    publishDir "${params.publish_dir}/short_centrifuge", mode:
'copy'

    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?

'https://depot.galaxyproject.org/singularity/centrifuge:1.0.4_be
ta--h9a82719_6' :
        'biocontainers/centrifuge:1.0.4_beta--h9a82719_6' }"

    input:
        path short_reads
        path database

    output:
        path 'short_cent'
        path 'short_cent_out.txt'

    script:
        """
        cat ${short_reads}/*.1_1.trim.fastq.gz
${short_reads}/*.2_1.trim.fastq.gz > short_reads_1.trim.fastq.gz
        cat   ${short_reads}/*.1_2.trim.fastq.gz
${short_reads}/*.2_2.trim.fastq.gz > short_reads_2.trim.fastq.gz
        cp ${database}/* \$PWD
        centrifuge -q -x p+h+v -1 short_reads_1.trim.fastq.gz -2
short_reads_2.trim.fastq.gz --threads 8 -S short_cent_out.txt --
report-file short_cent_out.tsv --min-hitlen 20

        gunzip short_reads_1.trim.fastq.gz

        gunzip short_reads_2.trim.fastq.gz

        mkdir short_cent
        mv short_reads_1.trim.fastq short_cent
        mv short_reads_2.trim.fastq short_cent
        """
}

process R_PROCESSING_SHORT {
    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?
        'https://depot.galaxyproject.org/singularity/r-
tidyverse%3A1.2.1' :
        'biocontainers/r-tidyverse%3A1.2.1' }"

    input:
        path rscript
        path short_inputs
        path short_cent
    output:
        path 'not_contam.txt'
        path 'short_cent'
    script:
    """
    cp ${rscript} \$PWD/centrifugeClean.R

    Rscript --vanilla centrifugeClean.R short_cent_out.txt
short_reads
    """
}

process SHORT_SEQTK {

    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?
        'https://depot.galaxyproject.org/singularity/seqtk:1.3--
h5bf99c6_3' :
        'biocontainers/seqtk:1.3--h5bf99c6_3' }"

    input:
        path sequences
        path not_contam

    output:
        path 'short_centrifuge_results'
    script:
    """
    seqtk subseq short_cent/short_reads_1.trim.fastq
not_contam.txt > clean_short_reads_1.trim.fastq
    gzip clean_short_reads_1.trim.fastq

    seqtk subseq short_cent/short_reads_2.trim.fastq
not_contam.txt > clean_short_reads_2.trim.fastq
    gzip clean_short_reads_2.trim.fastq


    mkdir short_centrifuge_results
    mv clean_short_reads_2.trim.fastq.gz
short_centrifuge_results
    mv clean_short_reads_1.trim.fastq.gz
short_centrifuge_results
    """
}
```

```
process HAPOG {
    publishDir "${params.publish_dir}/Hapog1", mode: 'copy'

    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?

'https://depot.galaxyproject.org/singularity/hapog:1.3.7--
py39hf95cd2a_0' :
        'biocontainers/hapog:1.3.7--py39hf95cd2a_0' }"

    input:
        path flye
        path flye_extra
        path centrifuge_out

    output:
        path 'hapog.fasta'

    script:
    """
    hapog --genome assembly.fasta --pe1
${centrifuge_out}/clean_short_reads_1.trim.fastq.gz --pe2
${centrifuge_out}/clean_short_reads_2.trim.fastq.gz -o polishing
-t 16 -u
    cp \$PWD/polishing/hapog_results/hapog.fasta
\$PWD/hapog.fasta
    """
}


process CHOPPER {
    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?

'https://depot.galaxyproject.org/singularity/chopper:0.7.0--
hdcf5f25_0' :
        'biocontainers/chopper:0.7.0--hdcf5f25_0' }"

    input:
        path long_reads
    output:
        path 'filtered_reads.fastq.gz'
    script:
    """
    gunzip -c ${long_reads} | chopper -q 20 -l 500 | gzip >
filtered_reads.fastq.gz
    """
}
```

```
process MINIMAP {
    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?

'https://depot.galaxyproject.org/singularity/minimap2:2.26--
he4a0461_2' :
        'biocontainers/minimap2:2.26--he4a0461_2' }"
    input:

        path genome
        path long_reads

    output:
        path 'aligned_long_reads.sam'

    script:
    """
    minimap2 -ax map-ont --secondary=no ${genome} ${long_reads}
> aligned_long_reads.sam
    """
}

process RACON1 {
    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?

'https://depot.galaxyproject.org/singularity/racon:1.5.0--
h7ff8a90_1' :
        'biocontainers/racon:1.5.0--h7ff8a90_1' }"
    input:
        path genome
        path long_reads
        path long_reads_sam


    output:
        path 'genome_SLpolished.fasta'

    script:
    """
    racon ${long_reads} ${long_reads_sam} ${genome} -t 10 >
genome_SLpolished.fasta
    """
}


process NTLINK {
    tag "ntLinking"
    publishDir "${params.publish_dir}/NTLink", mode: 'copy'

    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?

'https://depot.galaxyproject.org/singularity/ntlink:1.3.9--
py39hd65a603_2' :
        'biocontainers/ntlink:1.3.9--py39hd65a603_2' }"

    input:
        path nano_reads
        path genome
    output:
        path
'hapog_result.fasta.k32.w250.z1000.ntLink.5rounds.fa'

    script:

    """
    cp ${genome} \$PWD/hapog_result.fasta
    cp ${nano_reads} \$PWD/nanopore_raw.fastq.gz
    ntLink_rounds run_rounds target=hapog_result.fasta
reads=nanopore_raw.fastq.gz k=32 w=250 t=16 overlap=True
rounds=5
    """
}

process HAPOG2 {
    tag "HAPOGing the NTLINK"
    publishDir "${params.publish_dir}/Hapog2", mode: 'copy'

    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?

'https://depot.galaxyproject.org/singularity/hapog:1.3.7--
py39hf95cd2a_0' :
        'biocontainers/hapog:1.3.7--py39hf95cd2a_0' }"

    input:
        path ntlink
        path centrifuge_out

    output:
        path 'hapog.fasta'

    script:
    """
    hapog --genome
hapog_result.fasta.k32.w250.z1000.ntLink.5rounds.fa --pe1
${centrifuge_out}/clean_short_reads_1.trim.fastq.gz --pe2
${centrifuge_out}/clean_short_reads_2.trim.fastq.gz -o polishing
-t 16 -u
    mv \$PWD/polishing/hapog_results/hapog.fasta
\$PWD/hapog.fasta
    """
}

process SEQTK_RENAME {

    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?
        'https://depot.galaxyproject.org/singularity/seqtk:1.3--
h5bf99c6_3' :
        'biocontainers/seqtk:1.3--h5bf99c6_3' }"

    input:
        path genome
    output:
        path 'genome_renamed.fasta'
    script:
    """
    seqtk rename ${genome} n > genome_renamed.fasta
    """
}

process MINIMAP2 {
    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?

'https://depot.galaxyproject.org/singularity/minimap2:2.26--
he4a0461_2' :
        'biocontainers/minimap2:2.26--he4a0461_2' }"
    input:

        path genome
        path long_reads

    output:
        path 'aligned_long_reads.sam'
    script:
    """
    awk -F ' ' '/^>/{print \$1; next} 1' ${genome} >
genome_fixed_headers.fasta
    minimap2 -ax map-ont -N0 genome_fixed_headers.fasta
${long_reads} > aligned_long_reads.sam
```

```
    """
}

process RACON2 {
    publishDir "${params.publish_dir}/Racon2", mode: 'copy'

    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?

'https://depot.galaxyproject.org/singularity/racon:1.5.0--
h7ff8a90_1' :
        'biocontainers/racon:1.5.0--h7ff8a90_1' }"
    input:
        path genome
        path long_reads
        path long_reads_sam

    output:
        path 'genome_SL2polished.fasta'

    script:
    """
    racon ${long_reads} ${long_reads_sam} ${genome} -t 10 >
genome_SL2polished.fasta
    """
}

process RAGTAG {
    tag "Ragging and Tagging "
    publishDir "${params.publish_dir}", mode: 'copy'

    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?

'https://depot.galaxyproject.org/singularity/ragtag:2.1.0--
pyhb7b1952_0' :
        'biocontainers/ragtag:2.1.0--pyhb7b1952_0' }"

    input:
        path genome
        path ref1
    output:
        path 'Ragtag'
    script:
    """
    cp ${ref1} \$PWD/ref1.fasta
    cp ${genome} \$PWD/results.fasta
    ragtag.py scaffold -o Ragtag ref1.fasta results.fasta
    """
}

process BUSCOS {
    tag "Busco"
    publishDir "${params.publish_dir}/Busco", mode: 'copy'

    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?

'https://depot.galaxyproject.org/singularity/busco:5.5.0--
pyhdfd78af_0' :
        'biocontainers/busco:5.5.0--pyhdfd78af_0' }"
    input:
        path final_genome_path
        path ragtag_genome
        path busco
    output:
        path 'busco_genome'
        path 'busco_ragtag'
    script:
    """
    cp ${final_genome_path} \$PWD/results.fasta
    cp ${ragtag_genome}/ragtag.scaffold.fasta \$PWD/ragtag.fasta

    busco -m genome -i results.fasta -o busco_genome -l ${busco}

    busco -m genome -i ragtag.fasta -o busco_ragtag -l ${busco}
    """
}

process QUASTGENOME {
    tag "Quasting"

    publishDir "${params.publish_dir}", mode: 'copy'

    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?

'https://depot.galaxyproject.org/singularity/quast:5.2.0--
py39pl5321h2add14b_1' :
        'biocontainers/quast:5.2.0--py39pl5321h2add14b_1' }"

    input:
        path final_genome_path
        path quastGenomeGFF
        path quastGenomeFNA
    output:
        path 'quast_results'
    script:
    """
    cp ${quastGenomeGFF} \$PWD/input.gff
    cp ${quastGenomeFNA} \$PWD/input.fna
    cp ${final_genome_path} \$PWD/genome.fasta

    quast genome.fasta -r input.fna -g input.gff --large -t 10
    """
}

process QUASTRAGTAG {
    tag "Quasting"

    publishDir "${params.publish_dir}", mode: 'copy'

    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?

'https://depot.galaxyproject.org/singularity/quast:5.2.0--
py39pl5321h2add14b_1' :
        'biocontainers/quast:5.2.0--py39pl5321h2add14b_1' }"

    input:
        path ragtag_genome
        path quastGenomeGFF
        path quastGenomeFNA

    output:
        path 'ragtag_quast_results'
    script:
    """
    cp ${quastGenomeGFF} \$PWD/input.gff
    cp ${quastGenomeFNA} \$PWD/input.fna
    cp ${ragtag_genome}/ragtag.scaffold.fasta \$PWD/ragtag.fasta

    quast ragtag.fasta -r input.fna -g input.gff --large -t 10

    cp -r quast_results ragtag_quast_results
    """
}

process BAMCONVERSION {
    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?
```

```
        'https://depot.galaxyproject.org/singularity/mulled-v2-
e5d375990341c5aef3c9aff74f96f66f65375ef6:2cdf6bf1e92acbeb9b2834b
1c58754167173a410-0' :
        'biocontainers/mulled-v2-
e5d375990341c5aef3c9aff74f96f66f65375ef6:2cdf6bf1e92acbeb9b2834b
1c58754167173a410-0' }"

    input:
        path genome
        path centrifuge_out

    output:
        path 'short_reads_mapped2_sorted.bam'
        path 'short_reads_mapped2_sorted.bam.bai'

    script:
    """
    # index
    bwa-mem2 index -p mem2 ${genome}

    # map
    bwa-mem2 mem mem2 -t 40
${centrifuge_out}/clean_short_reads_1.trim.fastq.gz
${centrifuge_out}/clean_short_reads_2.trim.fastq.gz >
short_reads_mapped2.sam

    #convert to bam
    samtools view -bS short_reads_mapped2.sam >
short_reads_mapped2.bam

    #sort
    samtools sort short_reads_mapped2.bam >
short_reads_mapped2_sorted.bam

    #index
    samtools index short_reads_mapped2_sorted.bam
    """
}

process MOSDEPTH {
    tag "Mosdepthing"
    publishDir "${params.publish_dir}/Mosdepth", mode: 'copy'

    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?

'https://depot.galaxyproject.org/singularity/mosdepth:0.3.6--
hd299d5a_0' :
        'biocontainers/mosdepth:0.3.6--hd299d5a_0' }"

    input:
        path genome
        path index
    output:
        path 'mosdepth_result.*'
    script:

    """
    mosdepth -t 8 mosdepth_result ${genome}
    """
}

process MULTIQC {
    publishDir "${params.publish_dir}/MultiQC", mode: 'copy'

    container "${ workflow.containerEngine == 'singularity' &&
!task.ext.singularity_pull_docker_container ?

'https://depot.galaxyproject.org/singularity/multiqc:1.14--
pyhdfd78af_0' :
        'biocontainers/multiqc:1.14--pyhdfd78af_0' }"

    input:
        path pore_reads
        path pore_data
        path nano_ch
        path fastqc_ch
        path busco_rag
        path busco_gen
        path quast_gen_ch
        path quast_rag_ch
        path mosdepth_txt

    output:
        path 'multiqc_report.html'
    script:
        """
        multiqc . -d -dd 3
        """
}

workflow {
    // Stats
    nano_ch = NANOPLOT(longread_ch)
    fastqc_ch = FASTQC(shortreads1_ch, shortreads2_ch,
longread_ch)
```

```
    // Short Read prep
    fastp_short_ch = FASTP_SHORT(shortreads1_ch, shortreads2_ch)
    SHORT_CENTRIFUGE(FASTP_SHORT.out, phvDatabase)
    R_PROCESSING_SHORT(centrifugeRscript, SHORT_CENTRIFUGE.out)
    SHORT_SEQTK(R_PROCESSING_SHORT.out)

    // Long Read prep
    fastp_long_ch = FASTP_LONG(longread_ch)
    LONG_CENTRIFUGE(FASTP_LONG.out, phvDatabase)
    R_PROCESSING_LONG(centrifugeRscript, LONG_CENTRIFUGE.out)
    LONG_SEQTK(R_PROCESSING_LONG.out)
    CHOPPER(LONG_SEQTK.out)

    // Assembly
    FLYE(LONG_SEQTK.out)

    HAPOG(FLYE.out, SHORT_SEQTK.out)
    MINIMAP(HAPOG.out, CHOPPER.out)
    RACON1(HAPOG.out, CHOPPER.out, MINIMAP.out)

    NTLINK(longread_ch, RACON1.out)

    HAPOG2(NTLINK.out, SHORT_SEQTK.out)
    SEQTK_RENAME(HAPOG2.out)
    MINIMAP2(SEQTK_RENAME.out, CHOPPER.out)
    RACON2(SEQTK_RENAME.out, CHOPPER.out, MINIMAP2.out)

    RAGTAG(RACON2.out, ref1_ch)

    // Quality Checks
    busco_ch = BUSCOS(RACON2.out, RAGTAG.out, buscopath_ch)
    quast_gen_ch = QUASTGENOME(RACON2.out, quastpathGFF_ch,
quastpathFNA_ch)
    quast_rag_ch = QUASTRAGTAG(RAGTAG.out, quastpathGFF_ch,
quastpathFNA_ch)
    BAMCONVERSION(RACON2.out, SHORT_SEQTK.out)
    mosdepth_ch = MOSDEPTH(BAMCONVERSION.out)
    MULTIQC(fastp_long_ch, fastp_short_ch, nano_ch, fastqc_ch,
busco_ch, quast_gen_ch, quast_rag_ch,mosdepth_ch)
}
```

8.3 Centrifuge R-Script

```
#!/usr/bin/env Rscript
args = commandArgs(trailingOnly=TRUE)
library(dplyr)

f <- read.delim(file=args[1], header=TRUE)
print(nrow(f))
t <- f %>% filter(!seqID %in% "unclassified")
t <- t %>% mutate(V9 = hitLength/queryLength) %>%
arrange(desc(V9)) %>% filter(V9 > 0.50) %>% select(readID)
a <- anti_join(f,t) %>% select(readID) %>% distinct()
print(nrow(a))
write.table(a, file="not_contam.txt",col.names=FALSE
,row.names=FALSE,sep="\t", quote = FALSE)
```