MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

Detection and analysis of short variants in mitochondrial DNA

Master's Thesis

BC. VIKTÓRIA BENDÍKOVÁ

Brno, Fall 2023

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Detection and analysis of short variants in mitochondrial DNA

Master's Thesis

BC. VIKTÓRIA BENDÍKOVÁ

Advisor: doc. Ing. Matej Lexa, Ph.D.

Department of Machine Learning and Data Processing

Brno, Fall 2023


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Bc. Viktória Bendíková

Advisor: doc. Ing. Matej Lexa, Ph.D.

Acknowledgements

I would like to thank my advisor doc. Ing. Matej Lexa, Ph.D. and my consultant Mgr. Jan Kotrs for their valuable advice, prompt responses and guidance of this master thesis.

I would also like to express my appreciation to my family and partner for their endless support throughout the entire process of creating the thesis.

Abstract

Mitochondria are critical organelles of eukaryotic cells, which possess their own DNA. Variation in mitochondrial DNA is being increasingly linked to complex diseases, underlying the importance of developing strategies for their accurate identification and analysis.

This thesis addresses the challenges associated with identifying and analyzing variations in mitochondrial DNA. We develop a user-friendly and portable pipeline, which provides a high-level interface for variant detection built upon Genome Analysis Toolkit (GATK) guidelines. Additionally, the pipeline encompasses functionality for the analysis of mitochondrial DNA variants, necessary for their interpretation in the context of human health. The pipeline's adaptability is demonstrated by its deployment on the DNAnexus cloud-based platform.

Keywords

mitochondrial DNA, variant detection, GATK, pipeline, bioinformatics

Contents

List of Tables

List of Figures

Introduction

Mitochondria, critical organelles of eukaryotic cells, possess their own DNA with unique characteristics distinct from DNA in the nucleus. Variations in mitochondrial DNA play a major role in a variety of complex diseases including diabetes, cancer, or neurodegenerative disorders [1]. However, due to specific properties of mitochondrial DNA genetics, such as circularity of the mitochondrial genome or low heteroplasmy, the identification of variation in mitochondrial DNA is not a straightforward task, and adaptation of standard variant calling pipelines leads to inaccurate results.

The Genome Analysis Toolkit (GATK) developed by Broad Institute is an open-source set of tools considered an industry standard for calling genetic variation [2]. Additionally, GATK provides best practice recommendations tailored to numerous variant calling use cases. Recently, a set of best practices recommendations for calling short variants in mitochondrial DNA was released. These guidelines address challenges connected to mitochondrial variant calling and thus establish a solid foundation for the development of mitochondrial variant calling pipelines. However, GATK tools often contain a bewildering number of parameters, the best practices reference pipeline implementations are written in WDL (Workflow Description Language) [3] and optimized for running on the cloud-based platform Terra developed by Broad Institute. This setup can pose difficulties for users not familiar with the GATK ecosystem or WDL to adapt the pipelines and tools for their specific needs or execute them in different environments.

The main aim of this thesis is to explore specific aspects related to mitochondrial DNA variant identification and analysis. Building upon the robust foundation provided by GATK guidelines, we aim to construct an easy-to-use, flexible, and portable pipeline implementation. This pipeline will adhere to best practices for mitochondrial variant detection while hiding non-necessary details from the user, offering a high-level interface. Additionally, we intend to enhance the pipeline by incorporating crucial steps for the analysis and interpretation of mitochondrial variants. To demonstrate the portability of the created

pipeline, it will be adapted for use on DNAnexus, the cloud-based platform.

Chapter 1 briefly describes specific aspects linked to mitochondrial genetics. Chapter 2 introduces the main bioinformatical terms and concepts used in the process of pipeline implementation. The design and analysis of the individual stages of the pipeline, as well as end-to-end design, are discussed in Chapter 3. The remaining chapters are concerned with the implementation process and details, its deployment on the DNAnexus platform, and testing.

1 Biological background

This chapter provides a brief introduction to mitochondrial genetics, highlighting the key characteristics associated specifically with mitochondrial DNA. Furthermore, we explore the particulars of mutations in mitochondrial DNA and the associated consequences.

1.1 Human mitochondrial DNA

Mitochondria, also referred to as the powerhouses of the cell, are essential components of eukaryotic cells. Their primary function is to generate energy in the form of adenosine triphosphate (ATP) necessary to drive key metabolic processes within cells [4]. In the late 1960s, a rather controversial theory regarding the origin of mitochondria was published, proposing that mitochondria originated from the endosymbiotic relationship between alpha-proteobacterium and anaerobic protoeukaryotic host cell, which allowed the host cell to survive in an aerobic environment [5]. Although there is still a level of uncertainty and ongoing research regarding the exact bacterial origin of mitochondria [6], the theory of mitochondria being once an endosymbiotic bacteria, supported by extensive phylogenetic analyses, became globally accepted [4, 7, 8]. Due to their prokaryotic ancestry, mitochondria possess their own unique DNA (mtDNA) with several specific characteristics reminiscent of prokaryotic genomes and distinct from those of nuclear DNA (DNA that resides in the nucleus of the cell) [4].

1.1.1 Structure

The human mitochondrial DNA is a circular, double-stranded molecule with a length of 16 569 bp (base pairs) encoding for 2 ribosomal RNAs (rRNAs), 22 transfer RNAs (tRNAs) and 13 protein-coding genes that are part of the respiratory chain complexes. The two strands of mtDNA are termed heavy (H) and light (L) strands, with a majority of information being encoded on the heavy (H) strand. Unlike the nuclear genome, mitochondrial genes lack introns and noncoding intergenic sequences are absent, with the exception of a circa 1100 bp

long displacement loop (D-loop) region, a major control region for mtDNA expression [9].

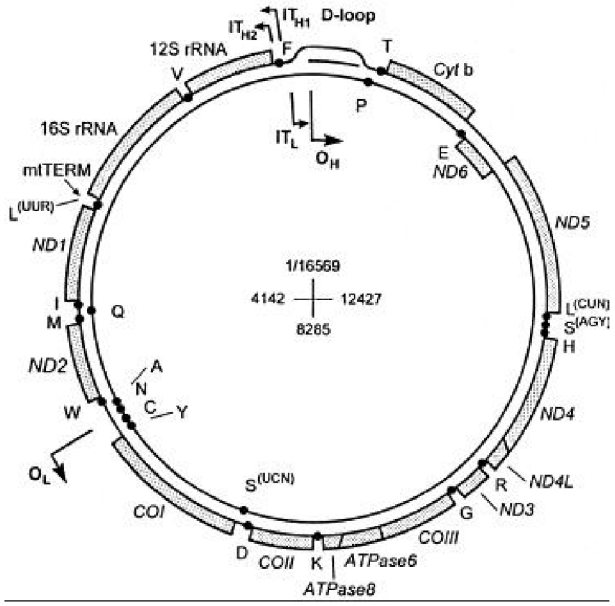


Figure 1.1: Human mitochondrial genome [9].

1.1.2 Key characteristics

The human (but also non-human) mtDNA follows a non-Mendelian inheritance pattern known as "*uniparental inheritance*", meaning that mtDNA is passed to the offspring only from one parental type, in contrast to nuclear DNA, which is inherited from both parents. In the context of mtDNA, we are speaking of maternal inheritance [10].

Depending on specific cell type, human cells contain from several hundred to ten thousand mtDNA copies per cell, accounting for less than one percent of total cellular DNA [11]. The important characteristic of mtDNA resulting from its multi-copy nature is the potential for the coexistence of multiple distinct mtDNA populations within a single cell, referred to as the state of heteroplasmy. On the other hand, homoplasmy refers to the presence of identical copies of the mitochondrial genome within a cell [12, 13].

Table 1.1: Main differences between nuclear DNA and mitochondrial DNA [15].

	mtDNA	nuclear DNA
Length	16 569 bp	3.3 billion bp
Organization	circular	linear
Copy number	several copies	one copy
Non-coding DNA	3 %	93 %
Inheritance	strictly maternal	both parents

There is a special phenomenon associated with the mitochondrial genome known as the nuclear mitochondrial DNA segment (NuMT). Nuclear mitochondrial segments are small fragments of mitochondrial DNA, that have been inserted into the nuclear genome. NuMTs are mostly no longer than 500 bp and usually originate from the control region of mitochondria [14].

For an overview, Table 1.1 shows the key differences between mitochondrial and nuclear DNA.

1.1.3 mtDNA haplogroups

Haplogroup refers to a group of similar haplotypes sharing specific mutations inherited from a common ancestor. From the phylogenetic point of view, a mitochondrial haplogroup represents a specific branch within a tree defined by variations in mitochondrial DNA, which are accumulated through maternal lineages. As a consequence, mtDNA represents an effective tool in a wide range of studies, including exploration of historical migration patterns, evolutionary development of human species, forensic science, and others [11, 16, 17].

Classification and assignment of haplogroups are currently captured by Phylotree [17], which provides a global phylogeny of human mtDNA variation and is widely accepted by the scientific community [18]. It contains over 5000 haplogroups and subhaplogroups labeled with the letters of the alphabet following specific nomenclature rules [11].

1.2 mtDNA variation

In comparison with nuclear DNA, mitochondrial DNA has a significantly higher mutation rate, as a consequence of several factors. Mitochondrial DNA is not protected by histone proteins and possesses limited repair mechanisms. Additionally, due to its proximity to respiratory chain complexes producing high quantities of reactive oxygen species (ROS), the mitochondrial genome is more vulnerable to oxidative damage [4, 19].

Another specific aspect of mitochondrial DNA variation arises from its multi-copy nature. As described above, there are several copies of mtDNA per cell, meaning the mtDNA mutations can exist in either homoplasmic or heteroplasmic state. In the homoplasmic state, all copies of mtDNA within a cell carry the same mutation. On the other hand, the mutations arising in mtDNA may affect just a certain proportion of mtDNA copies, allowing for the co-existence of normal and mutated mtDNA within a single cell, which adds significant complexity to the nature of mitochondrial diseases [12, 20].

Clinical manifestations specifically linked to mtDNA mutations include blindness, deafness, motor disorders, cardiovascular disease, muscle weakness, kidney dysfunction, and endocrine disorders such as diabetes. Furthermore, accumulated mtDNA mutations are being increasingly believed to play a role in ageing, neurodegeneration and cancer [1, 4, 19].

1.2.1 mtDNA variation types

The types of mutations that occur in mtDNA can be divided into two main categories: short point mutations and large-scale rearrangements [1]. In this thesis, our emphasis is on examining short variants (point mutations), thus we will refrain from discussing specifics of large rearrangements.

Speaking of short variants, we distinguish two main types. Single nucleotide variants (SNVs) are defined as an alteration of a single nucleotide within a genome sequence. In general, SNVs are the most common type of DNA variation [21]. Indels, short for insertion-deletion mutations, refer to insertions and deletions of sequence of nucleotides less than 1kb in length into DNA [22].

The biological impact of mitochondrial point mutations depends largely on the level of heteroplasmy and naturally on the genomic region where they occur (within protein-coding genes, tRNA or rRNA genes). The majority of mtDNA point mutations are highly recessive, meaning that the high proportion of mtDNA copies within the cell has to carry this mutation to observe a phenotype in cell [1, 11, 20].

2 Bioinformatical background

This chapter provides a conceptual overview of various steps included in the pipeline for the detection and analysis of mitochondrial variants. Section 2.1 highlights the specific bioinformatical challenges linked to mitochondrial DNA variant detection. Section 2.2 serves as an introduction to GATK and best practices guidelines. The concepts of individual phases of the pipeline are discussed in section 2.3. Since multiple file formats are used in the pipeline, section 2.4 provides a brief description of their syntax and purpose.

2.1 Challenges

The compact size of the mitochondrial genome compared to the nuclear genome might suggest that calling and analyzing mitochondrial variants is a straightforward task. However, as described in Chapter 1, the mitochondrial genome possesses several specifics that make the task of precise identification and analysis of mtDNA variants complicated from the bioinformatical perspective. This complexity requires adjustment of existing practices used for general variant identification and analysis.

Although the mitochondrial genome is circular, we have to use a linearized version to be able to apply existing alignment software. In the linearized mitochondrial reference sequence, the artificial breakpoint lies directly inside the control region (D-loop), making it challenging to sensitively detect variation in this highly variable region.

Furthermore, mapping of the mitochondrial genome is hindered by insertions of mitochondrial DNA into the nuclear genome (NuMTs). Due to their high sequence similarity, it is difficult to distinguish between reads belonging to autosomal regions and mitochondrial contigs, which introduces a certain level of false positivity in variant detection [14].

Lastly, the variants in mitochondria can exist at very low heteroplasmy levels and thus can be easily mistaken for sequencer noise [23, 24].

2.2 GATK best practices

Genome Analysis Toolkit (GATK), developed at Broad Institute, is the globally used open-source toolkit and industry standard for variant calling [2]. It includes a wide range of tools covering all stages of the variant calling process in next-generation sequencing (NGS) data. Additionally, GATK also provides step-by-step recommendations for variant discovery analysis in NGS data, tailored to specifics of variation of interest and leveraged technology. The GATK best practices pipelines are used at Broad Institute and are also provided in the form of workflow scripts implemented in WDL as a reference [25]. The GATK best practices for mitochondrial short variant discovery were first announced in 2019 and have undergone multiple changes since then. The key steps are described in the blog post [26] and the current reference WDL implementation is available on GitHub [1]. The proposed recommendations target the main challenges described in section 2.1. Through detailed analysis of best practices using available resources, we were able to understand the bioinformatical concepts and use them as the foundation of our pipeline implementation.

2.3 Pipeline

2.3.1 Preprocessing

Next-Generation Sequencing (NGS) is a massively parallel sequencing technology enabling high-throughput sequencing of DNA or RNA. NGS has a diverse range of clinical applications, including whole genome sequencing (WGS) or targeted sequencing [27]. WGS is a method for comprehensive analysis of an entire genome, including the mitochondrial genome, and allows for an effective identification of a wide range of genetic variations [28].

The first obligatory step in the variant discovery pipeline is preprocessing of raw sequencing reads. The basic preprocessing of raw reads includes alignment to the reference genome and multiple cleanup operations to produce analysis-ready alignment file [29]. Alignment

1. `https://github.com/broadinstitute/gatk/tree/master/scripts/mitochondria_m2_wdl`

refers to a process of assigning reads to the original positions in the reference genome.

In the mitochondrial detection pipeline, we assume that raw reads from the WGS experiment have been properly preprocessed and aligned to a whole reference genome, resulting in an analysis-ready BAM or CRAM file. As we are interested in the mitochondrial genome, the first step is to extract only reads aligned to the mitochondrial genome, usually marked as a separate chromosome (contig) in the whole reference genome. The extracted mitochondrial reads are consequently unaligned. The unalignment step is necessary since the following steps include realignment to the mitochondrial reference genome.

2.3.2 Alignment to mitochondrial reference genome

In the next stage of the pipeline, mitochondrial reads are realigned to the mitochondrial reference genome. As discussed in section 2.1, the conventional alignment software does not support alignment to circular genomes, which complicates the alignment stage in the mitochondrial pipeline. To address the limitation, a linearized form of mitochondrial reference is used in the alignment step, with an arbitral start and end position. However, in the circular representation, the arbitrarily designated start and end positions are in fact neighboring positions. As a consequence, reads that overlap in this position will not align well in the linearized sequence. Additionally, this artificial breakpoint lies exactly in the control region of mitochondria (D-loop), considered the most variable region of the mitochondrial genome, which spans ca. 500 bp before and after the first position of circular mitochondrial reference [30].

To overcome this issue and increase the sensitivity of calling variants in the control region, the double alignment strategy was proposed as a part of best practices [23]. The double alignment process includes alignment to mitochondrial reference, as well as shifted mitochondrial reference. The shifting of the mitochondrial reference can be viewed as cutting the reference circa in the middle (position 8000 bp), concatenating the first half of bases at the end of reference, making the 8001 position a new arbitrary start (see Figure 2.1). This allows reads origi-

nating from the control region to align accurately, increasing precision and sensitivity in the control region.

Additionally, to prepare alignments for variant calling, the usual procedure is to perform cleanup operations, such as detecting duplicate reads and sorting alignments. Marking duplicate reads is the process of identifying potential artifacts amongst reads, which are usually a result of DNA library construction using PCR.

Mitochondrial reference genomes

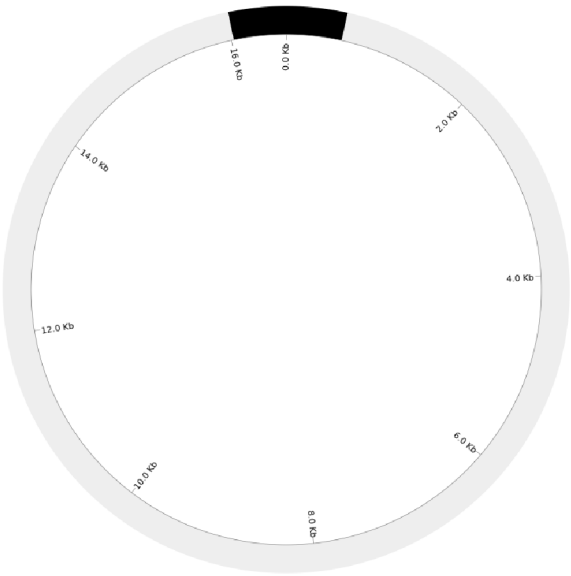
rCRS

The first reference sequence for human mitochondrial DNA was announced in 1981, generally known as the Cambridge Reference Sequence (CRS) [31]. However, several discrepancies have been discovered through further studies, which led to complete resequencing and finally publishing of a corrected version of CRS in 1999, officially marked as the revised CRS (rCRS) [32]. The rCRS is nowadays globally considered a standard reference sequence in human mtDNA research. The rCRS reference is freely accessible in the GenBank database under accession number NC_012920.1 [2]. This reference is included in most whole human reference genome assemblies (see Table 2.1).

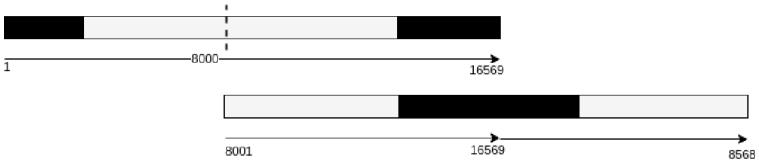
RSRS

In 2012, the replacement of rCRS with the newer Reconstructed Sapiens Reference Sequence (RSRS) was proposed by Behar et al. [33]. The RSRS keeps the same numbering system as rCRS, the main difference is that RSRS represents a reconstruction of the ancestral mitochondrial genome of Mitochondrial Eve, as opposed to rCRS, which is derived from an individual belonging to the European haplogroup. The main reason for the replacement of rCRS by RSRS was the reference bias of rCRS, which does not fully represent the genetic diversity. Although the RSRS has been leveraged in some studies, there was notable disagreement within the scientific community regarding the shift from using the rCRS to the RSRS as a standard in mtDNA research [34].

2. `https://www.ncbi.nlm.nih.gov/nuccore/251831106`



(**a**) Circular mitochondrial reference.



(**b**) Linearized canonical and shifted mitochondrial reference. The canonical mitochondrial reference is cut at position 8000, creating a shifted reference (below) with the start at position 8001 in original unshifted coordinates and the initial half being concatenated at the end.

Figure 2.1: Mitochondrial reference. Control region is highlighted (black).

The RSRS sequence is not deposited in GenBank but is freely accessible on the Phylotree website [3].

African Yoruba Sequence

An alternative Yoruba reference sequence (YRI) is a mtDNA genome sequence derived from an individual belonging to African haplogroup [35]. Due to different genome length (16571 bp), it uses a different position numbering system and additionally contains over 40 differences from rCRS. Although this reference was removed shortly after the announcement, it has been involved as a mitochondrial reference in earlier human genome release, the UCSC hg19 genome release, and adapted by major genotyping companies including Affymetrix and Illumina, which caused some major confusion [34, 36].

Table 2.1: Overview of mitochondrial references in human genome assemblies and their contig names.

Genome assembly	MT reference	MT contig name
GRCh38	rCRS	MT
GRCh37	rCRS	MT
hs37d5	rCRS	MT
hg38 (UCSC)	rCRS	chrM
hg19 (UCSC)	rCRS	chrMT
hg19 (UCSC)	YRI	chrM

2.3.3 Variant calling

Variant calling refers to a process of identifying differences between a reference genome and genome of an individual, or population of individuals. We denote these differences as variants, existing in multiple forms, such as single nucleotide variants (SNVs), short insertions and deletions (indels), or larger structural variants, as described in section 1.2. NGS technology allows for highly confident identification of genetic variants thanks to generating vast amounts of sequencing

3. `http://www.phylotree.org/resources/RSRS_annotated.htm`

data with high coverage depth (the number of reads that align at a given position) [37]. Notably, the compact size of the mitochondrial genome results in WGS offering extremely high depth coverage of the mitochondrial genome compared to the nuclear genome [38].

However, variant calling practices for nuclear genomes are not generally applicable to mitochondrial genomes. A particular feature of mitochondrial variants lowering the accuracy of typical variant calling procedure is the heteroplasmy, more specifically low heteroplasmy fractions. As described in the above chapters, mitochondrial variants can exist in a homoplasmic state, being present in all copies of mtDNA in a cell, as well as in a heteroplasmic state, where the variant is present only in a small fraction of mtDNA copies, making it challenging to distinguish them from technical errors (sequencer noise) or contamination [23, 24]. The heteroplasmy fraction of the variant is often referred to as variant allele frequency (VAF).

Since NGS data are more prone to specific types of artifactual variant calls, a crucial part of accurate variant calling assuring the reliability of results is precise filtering of raw variant calls to exclude common artifacts and other false-positive calls [39]. It is also common to perform normalization of variant calls to produce a standardized representation. This process involves left-aligning the variants, meaning shifting the variants to the leftmost position [40]. The common normalization step is also to split the sites containing more than 2 alternate alleles, so-called multiallelic sites, into separate variants.

2.3.4 Annotation

After identifying variants in the genome of interest, it is essential to assign relevant information to the variants to enable variant interpretation in a biological context and draw conclusions in a clinical setting. The process of assigning relevant biological information to the variant is generally known as variant annotation [41].

Variants can be annotated with different types of information, including variant context annotation, information about functional impact, associations with diseases, or variant frequencies in different populations. This information is usually stored in various annotation databases.

2.4 File formats

FASTA and FASTQ

FASTA is a simple text-based format developed to represent nucleotide and protein sequences. Entry in a FASTA file consists of a header marked with a ">" character providing descriptive information about the sequence (such as ID, name, etc.), followed by sequence data itself, where each letter corresponds to a nucleotide or amino acid in the case of a protein sequence. FASTA file can contain multiple sequences [42].

FASTQ file format is an extension of the FASTA format, which additionally contains information about quality scores for individual nucleotides in a sequence. The incorporation of quality information is essential for downstream analyses and thus makes the FASTQ a preferred format for storing sequencing data [42].

SAM/BAM/CRAM

SAM (Sequence Alignment Map) is a common tab-delimited text format designed to efficiently store read alignments against reference sequences, supporting all sequence types and alignment tools. The SAM format includes two main sections, an optional header section and an alignment section. All lines in the header section start with "@" character containing various metadata about the reference genome and alignment data. In the alignment section, each line has 11 mandatory fields which can be followed by a variable number of optional fields storing extra information from the platforms or alignment software [43].

However, SAM files tend to be relatively large and not space-efficient. BAM (Binary Alignment Map) format is a space-efficient version of SAM format, containing exactly the same information compressed by the BGZF library to achieve better space utilization [43].

To further reduce storage requirements, a CRAM (Compressed Reference-oriented Alignment Map), compressed columnar format was developed by the European Bioinformatics Institute. It builds upon the reference-based compression, where only differeneces between DNA sequence fragments and the reference genome they have

been aligned against are stored. This format is particularly useful for large-scale genomics projects [44].

uBAM

uBAM (unmapped BAM) is a special variant of BAM format where the read data does not contain mapping information, developed at Broad Institute for data management reasons. As indicated, the primary purpose was to keep all metadata of sequencing runs attached to the reads and completely avoid FASTQ files, which cannot store metadata associated with sequencing runs [45].

VCF

VCF (Variant Call Format) is a standard tab-delimited file format for storing genetic variation data, such as single nucleotide polymorphisms, structural variation, etc. It contains lines with metadata information (starting with "##" character), a header line defining 8 mandatory fields and optional fields followed by data lines, where each contains information about a single variant [46].

3 Analysis and Design

Thorough analysis and design is a crucial phase in the process of developing an application. In the context of our thesis, a large portion of the analysis phase involved a deep exploration of GATK best practices for mitochondrial variant detection, understanding the key concepts, and getting familiar with a variety of tools from GATK suite (bundles also Picard toolkit [47]). We used the blog post describing key steps [26] and a reference WDL implementation as the main resource. The design of the variant identification part in our pipeline is then based on these findings. Furthermore, we designed a part of the pipeline dedicated to the analysis and interpretation of the identified variants. In the section 3.1, we define the main requirements for the resulting application. Section 3.2 describes individual stages of the pipeline and discusses leveraged external tools. An end-to-end pipeline design is presented in section 3.3. Section 3.4 describes the chosen type of user interface. Finally, section 3.5 discusses the chosen strategy for dependency management and platform independence of the resulting tool.

3.1 Technical specification

To clearly define the functionality and features of the resulting software, we define a set of functional and non-functional requirements. The requirements originate from discussions with my consultant Mgr. Jan Kotrs. The tool is meant for general use by bioinformaticians and users with average IT skills.

3.1.1 Functional requirements

- the tool will be able to handle input WGS alignment files in BAM or CRAM format.

- the tool will provide functionality for mitochondrial variant detection from NGS data aligning with GATK best practices guidelines.

- beyond variant detection, the tool will provide functionality that enables variant analysis and interpretation, such as variant annotation and visualization.
- the user will be able to interact with the pipeline, having the possibility to run individual stages of the pipeline separately.
- the user will be able to customize analysis by tuning relevant parameters at various pipeline stages.

3.1.2 Non-functional requirements

- since the tool will perform a complex workflow consisting of many stages, it should provide extensive documentation for the end users.
- the installation process of the tool should be as straightforward as possible and well-documented in an installation guide.
- the third-party tools integrated in the pipeline should be well tested and maintained.
- the tool should be platform-independent.
- the tool should provide an interface, which would enable to customize and scale the pipeline using the workflow manager of choice.

3.2 Pipeline stages

3.2.1 Preprocessing

As the input, the preprocessing stage will accept WGS alignment file in BAM or CRAM format and will perform two preprocessing steps. The first step performs extraction of the reads mapped to the mitochondrial genome from the input alignment file. This extraction is accomplished through the utilization of the GATK PrintReads tool, specifically designed for extracting reads from alignment format files that meet specified criteria. The extracted reads are then unaligned

using the Picard tool RevertSam capable of removing alignment information and producing an unmapped BAM (uBAM) file. If the input alignment file is in CRAM format, the reference genome is additionally required to perform read extraction. Figure 3.1 depicts the flow of the preprocessing stage.

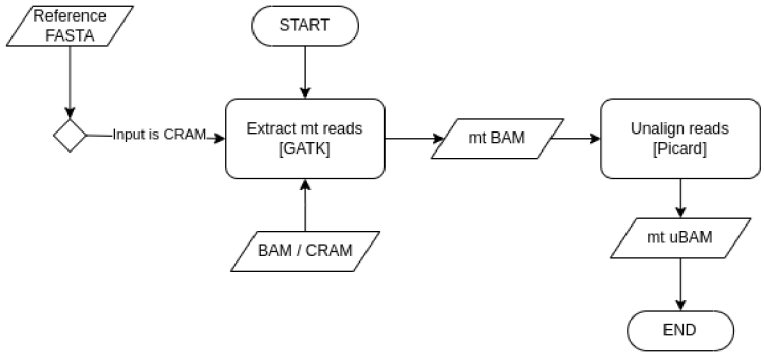


Figure 3.1: Preprocessing stage.

3.2.2 Alignment to mitochondrial reference genome

The necessary step prior to performing alignment to the mitochondrial genome is the conversion of input unaligned mitochondrial reads from uBAM to FASTQ format, which is a required format for most of the aligners. The Picard SamToFastq tool is leveraged for this purpose.

Regarding the alignment process, BWA (short for Burrows-Wheeler Aligner) [48] is a standard tool of choice for the alignment of short-read sequences within GATK pipelines. BWA consists of three algorithms: BWA-backtrack, BWA-SW and BWA-MEM, each tailored to a specific read length. The BWA-MEM algorithm is the latest, most commonly used for high-quality alignments in general-purpose applications. For our pipeline, we opted for an optimized version of the BWA-MEM algorithm, BWA-MEM2, which produces alignment results identical to BWA-MEM while being 1.3-3.1x faster [49].

During to conversion of uBAM to FASTQ, certain useful information from the uBAM is lost. In order to preserve this information and pass it to downstream steps, the unmapped BAM is merged with the SAM file produced by the aligner, and a new BAM file is created, which

contains information from uBAM, as well as all alignment information. This is the purpose of the Picard MergeBamAlignment tool.

Finally, the last alignment phase involves marking duplicates and sorting, in order to prepare the BAM file for the variant calling stage. According to best practices, the Picard MarkDuplicates and SortSam tools are used for this purpose. For better performance reasons, we chose to use GATK MarkDuplicatesSpark tool, a Spark implementation of Picard MarkDuplicates, that additionally performs sorting and can be run locally on a single machine leveraging core parallelism. Even without access to a Spark cluster, it runs faster than MarkDuplicates and SortSam while producing identical results. To get a better picture of the alignment stage, Figure 3.2 depicts the alignment process.

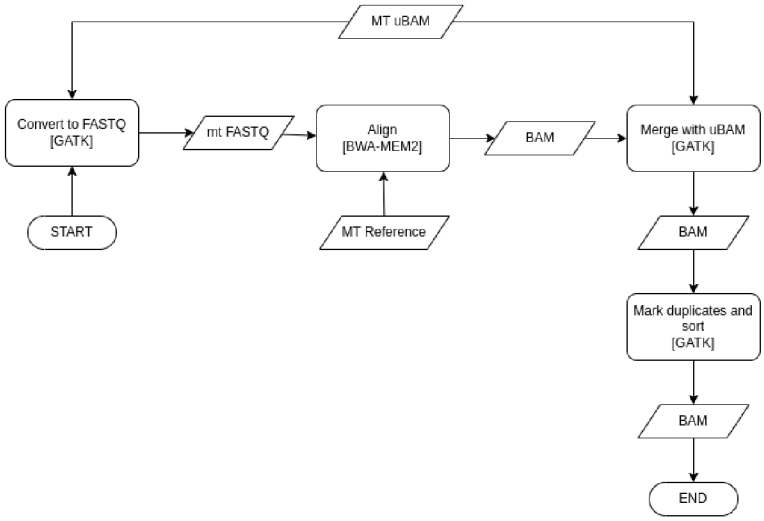


Figure 3.2: Alignment stage.

3.2.3 Variant calling

The mitochondrial variants are called using GATK Mutect2, a variant caller from the GATK toolkit, which was originally designed for calling short somatic variants in tumor samples [50]. However, it includes a special mode developed to call variants in mitochondrial genomes, which automatically tunes specific filters to increase the sensitivity of

calling at high sequencing depths and report variants at all levels of heteroplasmy [23].

3.2.4 Merging variant calls

After calling variants separately for control and non-control regions of the mitochondrial genome, the variant calls from both regions have to be merged. The variants in the control region were called against shifted mitochondrial reference and thus have to be shifted back to coordinates of canonical reference sequence. Picard LifotverVcf tool provides the desired functionality for lifting over a VCF file from one reference build to another. To merge VCF and shifted-back VCF, the Picard MergeVcfs tool is used. Additionally, VCF stats files required for the postprocessing stage are merged using MergeMutectStats. Figure 3.3 shows the process of the merging stage.

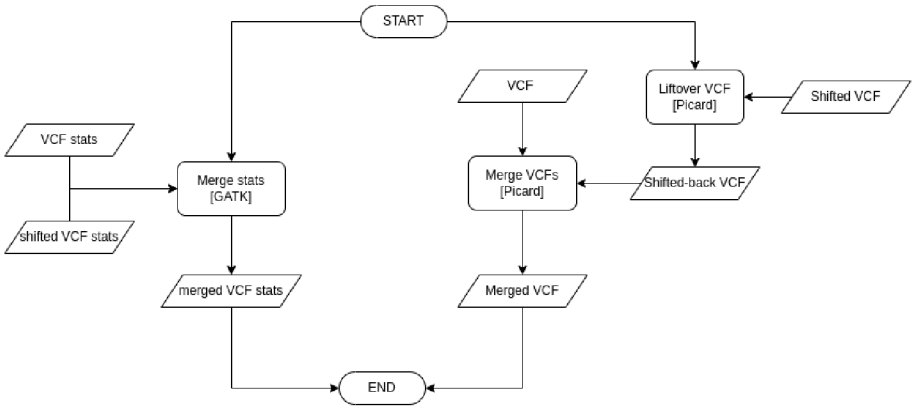


Figure 3.3: Merging stage.

3.2.5 Postprocessing

Raw variant calls have to undergo a series of postprocessing steps, which include filtering and normalization in order to eliminate potential false-positive calls and output variants in standard representation. The initial filtering phase includes filtering variants based on multiple specified parameters using the GATK FilterMutectCalls tool specifically designed for filtering raw Mutect2 calls. Optionally, the

variants are filtered based on estimated contamination level. To estimate the level of contamination in mitochondrial DNA samples, we utilize Haplocheck [51].To the best of our knowledge, this is the only tool specifically tailored for detecting contamination in mitochondrial DNA. The next level of filters eliminates common artifacts, i.e. variants overlapping known artifact-prone mitochondrial sites. The last filtering phase involves filtering out potential NuMTs. Based on the GATK best practices documentation [26], the NuMTs can be filtered by specifying an additional parameter in the FilterMutectCalls tool. However, this tool's option is not available in the latest version of the GATK toolkit. To overcome this issue, we use GATK NuMTFilterTool, which is specifically designed to filter out NuMTs.

Following filtering, the individual variant calls are normalized. This process involves left-aligning the variants and splitting multiallelic sites. For this purpose, the GATK LeftAlignAndTrimVariant tool is used.

As the final step, the variants not passing specified filters (do not include "*PASS*" in the FILTER field of the VCF file) are removed from the final VCF file using the GATK SelectVariants tool. The whole postprocessing flow is shown in Figure 3.4.

3.2.6 Calculating coverage

To be able to assess the reliability of resulting variant calls, it is important to explore per-base coverage [38]. Based on the coverage, one can decide if there is enough evidence, i.e. enough reads covering specific position in the genome to consider a call at the position reliable. There are several widely used tools for calculating coverage, including samtools depth [43], BEDTools genomecov [52], sambamba [53] and the most recently published tool mosdepth [54]. All tools are able to calculate per-base coverage, the difference lies in the performance of individual tools. Out of the mentioned tools, mosdepth has the best performance being twice as fast as the fastest samtools depth [54], which is the reason for choosing mosdepth to perform per-base coverage calculation.

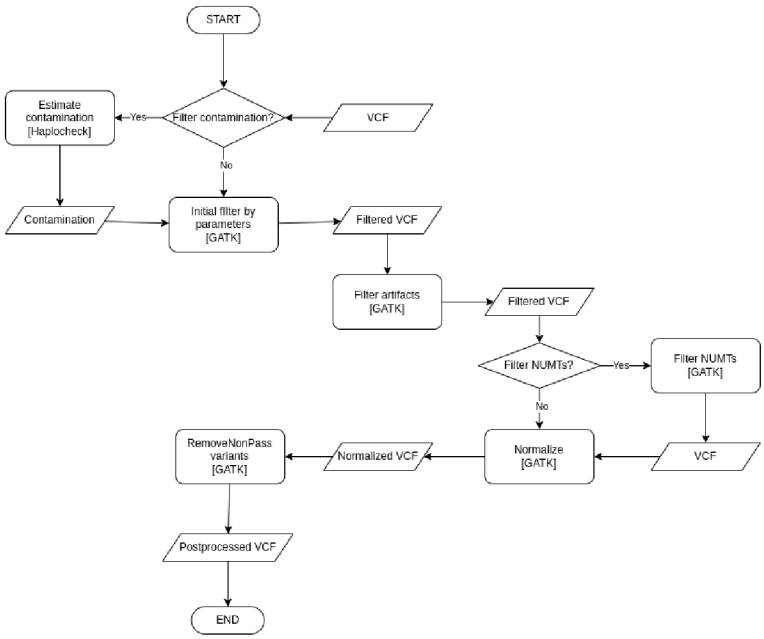


Figure 3.4: Postprocessing stage.

3.2.7 Identifying haplogroups

There are currently several tools capable of performing the identification of haplogroups based on mtDNA variants. As we want to identify haplogroups from detected variants, we discarded all tools that do not accept VCF as an input file. Additionally, after discarding no longer supported tools and tools that cannot be run locally, we were left with a choice between Haplocheck [51], Haplogrep [55] and Haplogrouper [56]. Based on an extensive benchmarking study, Haplocheck emerges as the most suitable haplogroup identification tool for WGS data [57]. Haplocheck has more applications beyond just haplogroup identification and uses Haplogrep as an underlying tool for haplogroup identification. However, it still uses Haplogrep 2 [58], which is a no longer maintained version of Haplogrep and has been replaced with Haplogrep 3 [59]. Haplogrep 3 additionally supports different mitochondrial phylogenies provided for rCRS and RSRS mitochondrial references as opposed to its previous version, which supports only rCRS. Based on the mentioned reasons, we chose Haplogrep 3 to perform haplogroup classification.

3.2.8 Annotation

We designed a custom offline annotation module, which integrates information from multiple general annotation sources, as well as databases storing solely mitochondrial data. Usually, annotation tends to be a computationally demanding step, as it involves searching for information in large annotation databases. We took advantage of the compact size of the mitochondrial genome and prepared custom annotation databases containing only mitochondrial data. Additionally, we leveraged available databases specifically dedicated to mitochondrial information.

There are many of biologically relevant information that can be attached to the variants. We chose to extensively annotate mitochondrial variants with functional information, population frequencies, site context information, in-silico pathogenicity predictions, and disease associations. Additionally, we annotate variants to be either homoplasmic or heteroplasmic based on the provided minimum homoplasmy threshold. Figure 3.5 shows a designed flow of the annotation stage.

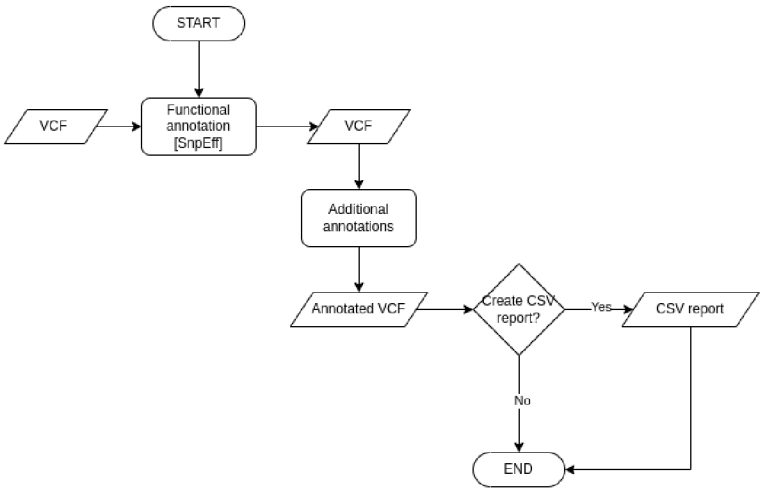


Figure 3.5: Annotation stage.

Heteroplasmy/homoplasmy annotation

In order to be able to determine if the mitochondrial variant is homoplasmic or heteroplasmic, the minimum homoplasmy threshold

has to be set. Based on the threshold, which is set to 95% by default (as per gnomad [23]), we annotate all variants with variant allele frequency above this threshold as homoplasmic, otherwise the variant is considered heteroplasmic.

Functional annotation

Functional annotation involves predictions of the functional effects of the variant and their significance on the gene functions. Considering two main functional categories, the variant can be considered synonymous, meaning they do not alter the amino acid sequence of the protein, or non-synonymous, meaning that mutation results in changes in the protein sequence. The most widely used general tools designed for functional annotation of variants are ANNOVAR [60], VEP [61] and SnpEff [62]. The main factor guiding our selection of a tool for mitochondrial functional annotation was the possibility to efficiently build a custom database focused exclusively on mitochondria and use it for annotation. For ANNOVAR, we did not find a straightforward way to use a custom prebuilt database. Both SnpEff and VEP provide an easy way of annotation with custom databases in GFF/GTF/GenBank file format. Since VEP is written in Perl and depends on additional Perl packages, it would add an additional layer of complexity to dependency management of our tool. As we already include Java in our dependencies (due to the GATK toolkit), we decided to perform functional annotation with java-based SnpEff.

Population frequencies

Population frequency data play a critical role in assessing the pathogenicity of genetic variants. Variants that are common in the general population are less likely to be pathogenic, whereas rare variants may have more significance [63]. We include population annotations from the gnomAD database, a publicly available large-scale genomic database that aggregates information about genetic variants from a diverse set of human exome and genome sequencing projects [64]. The mitochondrial variants for 56,434 whole genome samples were first included in the v3.1 database release [23]. The population frequency annotation generally includes overall allele counts (how many times

was allele observed in a specific population) and allele frequencies (proportion of a particular allele in a population). For mitochondrial variants, the gnomAD database does not offer overall allele counts and frequencies but provides them separately for homoplasmic and heteroplasmic variants. The minimum threshold for considering variant to be homoplasmic is set to 95% in the gnomAD callset.

Pathogenicity predictions

Another relevant source for assessing the pathogenicity of variants is in-silico pathogenicity predictions. Specifically for variants located in the tRNA locus of the mitochondrial genome, two major resources are being used to estimate pathogenicity: MitoTIP and PON-mt-trna. MitoTIP is an in-silico tool designed to predict the pathogenicity of mitochondrial tRNA variants [65]. The prediction is based on database frequencies, type of the nucleotide change, and conservation scores. PON-mt-trna is a method combining machine-learning and evidence-based likelihoods of pathogenicity to predict the probability of tRNA variants pathogenicity [66]. In our annotation, we use freely available precomputed scores from both tools.

For the non-synonymous mitochondrial variants, we include precomputed predictions performed by SIFT, a commonly used general tool for prediction of the influence of amino acid change on protein function [67].

Phenotype annotations

To build a comprehensive picture of the variant's impact on human health, we add annotations that connect variation with observed phenotypes. MITOMAP database aggregates a wide range of information on human mitochondrial DNA variation, including reported associations of mitochondrial variants with various diseases [68]. We complete the phenotype annotation with disease associations of mitochondrial variants from a widely used, general phenotype database ClinVar [69].

Site annotations

Variant annotations are completed with annotations specific to the mitochondrial sites, which include general annotations and conservation scores. General annotations are composed of a locus (position of the variant within a genome) and its biotype (classification of a particular locus, in the context of mtDNA can be either protein-coding, rRNA, tRNA, or regulatory).

Conservation scores are a numerical representation of the evolutionary conservation of a particular region within a genome. In the context of variant interpretation, the variants occurring in the highly conserved regions are more likely to have functional consequences as highly conserved regions are associated with functional importance. Conservation scores can be computed by multiple tools and are often available as precomputed scores. In our annotation, we include precomputed conservation scores by PhyloP [70] and PhastCons [71] for human genome (hg38) calculated from multiple alignments with other 99 vertebrate species [72].

3.2.9 Visualization

To get a comprehensive overview of identified variants, we designed an interactive graphical representation. We chose to implement visualization using plotly [73], a Python library providing an interface for efficient creation of interactive visualizations.

We opted for circular layout for visualization to align with the circular nature of the mitochondrial genome. Key features of the visual representation will include the precise positioning of variants along the circular mitochondrial genome, providing a spatial context for identified variants. This can help with identifying patterns, clusters, or specific regions with a higher density of variants. For each identified variant, essential information will be included, specifically details about the reference and alternate alleles, to allow users to quickly understand the nature of the genetic change (whether it involves a substitution, insertion, or deletion). Furthermore, the variant allele frequency, also known as heteroplasmy fraction, will be included in the visualization, since it provides valuable information about the abundance of the variant within the mitochondrial DNA population.

Optionally, the visual representation will be able to incorporate per-base coverage information, to offer a comprehensive view of the sequencing depth at each position. This feature will allow users to quickly assess the reliability and confidence of identified variant calls.

3.3 End-to-end pipeline design

Based on the conducted analysis, the end-to-end process of the pipeline for single sample is designed, incorporating all stages for variant detection, as well as variant analysis. Figure 3.6 shows the flow of the pipeline.

As the first step, the input alignment file in either BAM or CRAM format is run through the preprocessing stage, generating unaligned mitochondrial reads in uBAM format. Consequently, the double alignment strategy is performed, which involves the alignment of mitochondrial reads to mitochondrial reference, as well as shifted mitochondrial reference, producing two BAM files. The variants are then called separately for the control region, using alignment to shifted mitochondrial reference, and non-control region using the alignment to canonical reference. The two variant callsets are merged and run through the postprocessing stage, producing filtered and normalized VCF file.

The postprocessed VCF file enters the variant analysis phase of the pipeline. The VCF is annotated and optionally, a CSV report with annotations is produced. Variants are visualized and haplogroups are identified from the postprocessed VCF file. The per-base coverage is calculated and combined from canonical and shifted alignments, and optionally included in the final variant visualization plot.

The main outputs of the pipeline include both alignment files, postprocessed VCF file, haplogroup report, annotated VCF file, optionally CSV annotation report, and visualization in HTML (and optionally PNG) format.

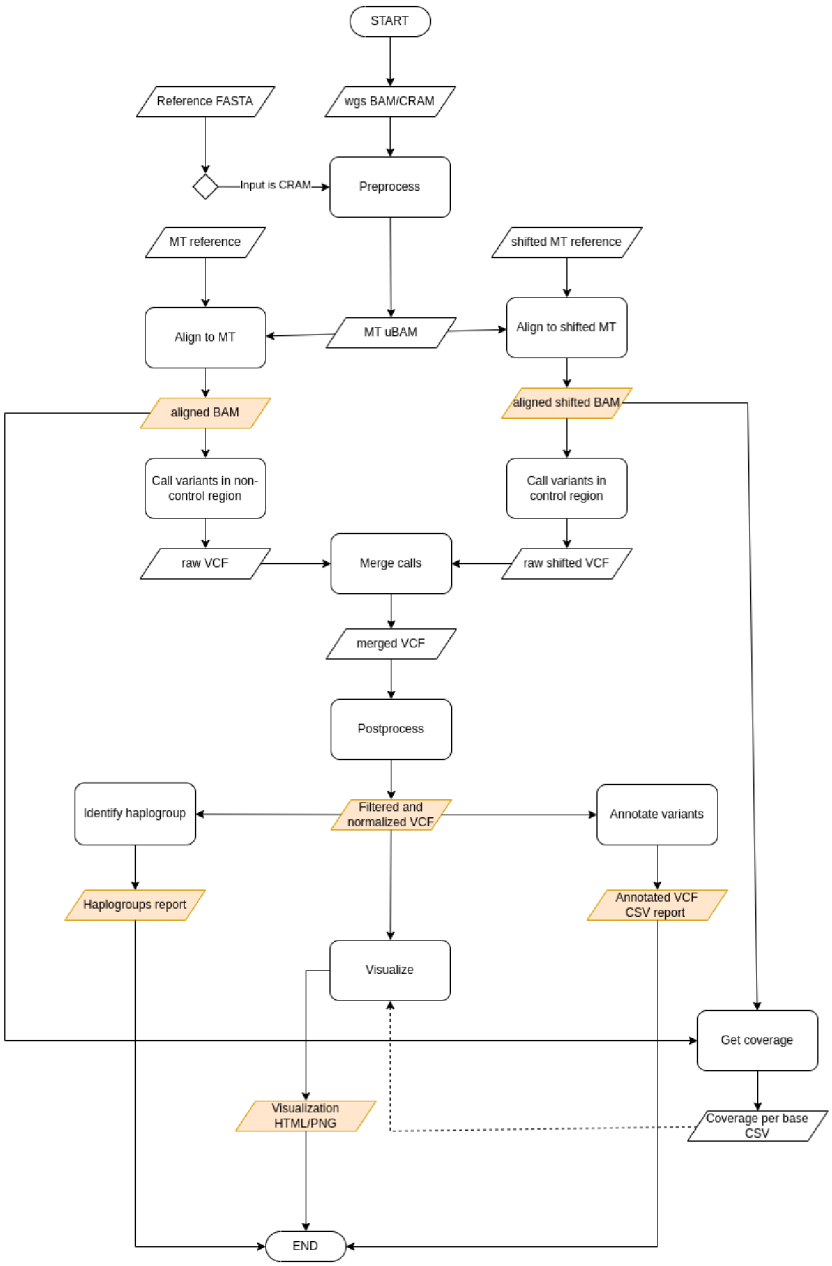


Figure 3.6: End-to-end pipeline design. The main outputs are colored (orange).

3.4 User Interface

As our tool is meant for users with average IT skills, running the pipeline and individual pipeline stages will be controlled through a text-based command line interface. This solution is simple and provides greater flexibility in terms of potential extensions and modifications. The user can customize individual pipeline stages, as well as the whole pipeline by tuning relevant parameters. It would also enable to seamlessly incorporate pipeline stages of interest in other pipelines, or build scalable pipelines using the workflow manager of choice.

3.5 Virtualization

Virtualization is a common strategy to deliver platform-independent software and efficiently manage its dependencies. It is an abstract, virtual environment, that enables an efficient utilization and sharing of physical resources without exposing user to underlying implementation details. This abstraction allows multiple isolated virtual instances to run independently on a single physical machine while optimizing resource allocation [74].

Conceptually, we distinguish between two main types of virtualization: hypervisor-based and container-based virtualization. Hypervisor-based virtualization involves virtual machines with their own guest operating system and kernel, creating strong isolation from the host machine. Compared to hypervisor-based virtualization, container-based virtualization represents a more lightweight form of virtualization. Containerization involves the utilization of the host operating system, as well as the kernel. On one hand, this allows for better performance and scalability as fewer resources are needed, on the other hand, it creates security risks [75]. In the context of scientific workflows, the usual choice is a lightweight container-based virtualization, as it enables to create isolated complex runtime environments with large amounts of dependencies and therefore allows for an easy distribution of software [76].

Therefore, we chose to use container-based virtualization strategy, concretely Docker to satisfy the requirement of an easy installation process, efficient dependency management and platform-independence.

4 Implementation

This chapter deals with concrete implementation details of the resulting pipeline based on the design presented in the previous chapter. We implemented the pipeline as an application called *mitopy*, controlled by a command line interface in Python, a general programming language popular within the bioinformatics community [77]. Each stage of the pipeline is implemented as a separate module, to allow for flexibility, easier maintenance, and testability. Furthermore, extensive documentation was created using Sphinx [78] and is hosted on ReadTheDocs [1]. Application *mitopy* is open-source, available on GitHub [2] under MIT license.

Section 4.1 describes the preparation process of mitochondrial reference genomes. Implementation of individual pipeline stages and concrete use of specialized tools is described in section 4.2. Section 4.3 discusses dependency management and creating a container-based environment using Docker. In section 4.4, we briefly describe the application of DevOps principles in our software. We demonstrate the use of the application to build a scalable solution using Nextflow as the workflow manager of choice in section 4.5. Lastly, we describe the deployment process of the nextflow-based pipeline implementation to DNAnexus, a cloud-based platform in section 4.6.

4.1 Preparation of mitochondrial references

The rCRS reference genome sequence was obtained from the NCBI database under the accession number NC_012920.1. The RSRS reference FASTA sequence was obtained from the PhyloTree website. The shifted mitochondrial references for both mitochondrial genomes were obtained using the GATK ShiftFasta tool:

```
gatk ShiftFasta -R <FASTA> -O <SHIFTED_FASTA>
    --shift-back-output <SHIFT_BACK_CHAIN>
    --shift-offset-list 8000
```

1. https://mitopy.readthedocs.io
2. https://github.com/bendda/mitopy

We shift the original reference FASTA files by 8000 bases (specified by `--shift-offset-list` option, meaning the 8001 is the first base in the shifted FASTA. The tool also outputs a `SHIFT_BACK_CHAIN` file, which contains information needed to shift FASTA back to its original coordinates.

Both references and their shifted versions were prepared for the analysis, generating corresponding index files (.fai) using *samtools faidx* and dictionary files (.dict) using the Picard CreateSequenceDictionary tool. Additionally, the index files required for the alignment stage performed by the bwa-mem2 aligner were generated using the following command:

```
bwa-mem2 index <REFERENCE>
```

Prepared reference files, both original and shifted, along with their indexes and dictionaries, are included in *mitopy*.

4.2 Pipeline stages

4.2.1 Preprocessing

The preprocessing module takes an input alignment file in either BAM or CRAM format and produces mitochondrial reads stored in unmapped BAM format. It consists of two steps as specified in best practices, implemented using GATK tools. The input alignment file should be sorted by coordinate and indexed (the index file should be present). If the input file is in CRAM format, the reference genome in FASTA format, as well as the corresponding index and dictionary file is needed. Additionally, the mitochondrial contig name should be specified based on the reference genome the reads were aligned against, as mitochondrial contig name differs across human genome assemblies.

Prior to executing the preprocessing steps, *mitopy* verifies if the input alignment file is sorted based on its header and will sort the BAM file if it is found to be unsorted. If the index file is not explicitly specified, *mitopy* will check if the corresponding index file is present in the alignment's file directory and will create an index for the BAM file if one is not found. If the mitochondrial contig name is not explicitly specified by the user, it will be dynamically determined from the

header of the input alignment file. If the input is a CRAM file, the existence of reference genome fasta, its index (.fai), and dictionary (.dict) file will be verified. If not found, *mitopy* will create an index using `pysam.faidx()` and FASTA dictionary using a Picard CreateSequenceDictionary tool. For creating index files, sorting, and detecting the name of the mitochondrial contig from input BAM/CRAM files, we use *pysam*[3], a Python module for manipulating SAM/BAM/CRAM files.

First, the input alignment file is subsetted to include only reads mapped to the mitochondrial genome using GATK PrintReads tool with the following options:

```
gatk PrintReads -I <BAM> -O <OUT_BAM>
    -R <REFERENCE> -L <MITO_CONTIG>
    --read-filter MateOnSameContigOrNoMappedMateReadFilter
    --read-filter MateUnmappedAndUnmappedReadFilter
```

To retain only mitochondrial reads, the mitochondrial contig name is passed to the genomic interval option `-L`. Two read filters are specified according to best practices.

In the next step, the subsetted BAM file is unaligned to remove all alignment information while retaining the recalibrated base qualities and original alignment tags using Picard RevertSam:

```
gatk RevertSam -I <BAM> -O <OUT_BAM>
    --VALIDATION_STRINGENCY LENIENT
    --ATTRIBUTE_TO_CLEAR FT
    --ATTRIBUTE_TO_CLEAR CO
    --RESTORE_ORIGINAL_QUALITIES false
```

By setting `--RESTORE_ORIGINAL_QUALITIES` to false, the recalibrated base qualities are retained. Optional alignment tags to remove are specified through the `--ATTRIBUTE_TO_CLEAR` option, in this case, FT and CO, as they will be reused by downstream processes. Specifying `--VALIDATION_STRINGENCY` to LENIENT allows for more flexibility by tolerating certain input alignment file issues and suppressing unnecessary warnings and errors, improving overall performance.

3. `https://github.com/pysam-developers/pysam`

4.2.2 Alignment to mitochondrial genome

The alignment module takes an unaligned BAM file with mitochondrial reads and aligns them against the mitochondrial genome. *mitopy* supports alignment against rCRS and RSRS mitochondrial reference genomes. Furthermore, it includes the option to align against shifted mitochondrial references, to support the double alignment strategy described in best practices.

As the bwa-mem2 aligner requires input reads to be in FASTQ format, the first step is to convert reads from uBAM format to FASTQ format using GATK SamToFastq tool:

```
gatk SamToFastq -I <BAM> -F <OUT_FQ>
    --INTERLEAVE true
    --INCLUDE_NON_PF_READS true
```

The `--INTERLEAVE` flag defines a strategy of dealing with paired reads, and in our case is set to true to generate an interleaved FASTQ file, where each line contains an identifier /1 or /2 to describe which end the read came from. To include reads that do not pass quality filtering, the `--INCLUDE_NON_PF_READS` is set to true.

The reads in FASTQ format are passed to the bwa-mem2 aligner and alignment to the mitochondrial genome is performed with the following command:

```
bwa-mem2 mem -p -v 3 -t <THREADS>
    -K 100000000 -Y
    -o <OUT_SAM> <READS_FASTQ>
```

Setting the `-p` flag indicates that input reads are in interleaved paired-end FASTQ format, `-K` option specifies a seed to achieve deterministic alignment results. The `-Y` flag is set to force a soft clipping strategy instead of default hard clipping for supplementary alignments in order to preserve information about potential alternative mappings.

The unmapped BAM is merged with the SAM file produced by the aligner using the Picard MergeBamAlignment tool with the following options:

```
gatk MergeBamAlignment -ALIGNED <BAM> -UNMAPPED <uBAM>
    -O <OUT_BAM> -R <REFERENCE>
    --VALIDATION_STRINGENCY SILENT
    --EXPECTED_ORIENTATIONS FR
    --ATTRIBUTES_TO_RETAIN XO
    --ATTRIBUTES_TO_REMOVE NM
    --ATTRIBUTES_TO_REMOVE MD
    --SORT_ORDER queryname
    --CLIP_ADAPTERS false
    --MAX_RECORDS_IN_RAM 2000000
    --MAX_INSERTIONS_OR_DELETIONS -1
    --PRIMARY_ALIGNMENT_STRATEGY MostDistant
    --UNMAPPED_READ_STRATEGY COPY_TO_TAG
    --ALIGNER_PROPER_PAIR_FLAGS true
    --UNMAP_CONTAMINANT_READS true
    --ADD_PG_TAG_TO_READS false
```

All options are set following the GATK best practices guidelines, with and exception of `--SORT_ORDER` option, which was changed from "*unsorted*" to "*queryname*". This was done out of performance reasons since this step is followed by the GATK MarkDuplicatesSpark tool, which is optimized to run on alignments sorted by queryname.

Finally, the duplicate reads are marked and the BAM file is sorted by coordinate using GATK MarkDuplicatesSpark tool in local mode:

```
gatk MarkDuplicatesSpark -I <BAM> -O <OUT_BAM>
    -M <METRICS>
    --optical-duplicate-pixel-distance 2500
    --create-output-bam-splitting-index false
    --spark-runner LOCAL
    --spark-master local[<CORES>]
```

The `--optical-duplicate-pixel-distance`, a maximum distance between two duplicate groups used to determine if they can be considered optical duplicates is set to 2500, which is considered an optimal threshold for patterned flowcell models. The Spark implementation supports running the tool on a local machine using multiple cores, which is specified by setting `--spark-runner` to LOCAL and specifying the number of cores through `--spark-master` option.

4.2.3 Variant calling

The mitochondrial variants are called using *Mutect2* variant caller from GATK toolkit in the mitochondrial mode specified by `--mitochondria-mode` flag:

```
gatk Mutect2 -I <BAM> -O <OUT_BAM>
    -R <REFERENCE> -L <INTERVAL>
    --read-filter MateOnSameContigOrNoMappedMateReadFilter
    --read-filter MateUnmappedAndUnmappedReadFilter
    --annotation StrandBiasBySample
    --mitochondria-mode
    --max-reads-per-alignment-start 75
    --max-mnp-distance 0
```

The *mitopy* variant calling module provides two modes of variant calling based on the double alignment strategy. When calling variants in non-control region, the BAM aligned to the canonical mitochondrial reference is used alongside with canonical reference. The interval option `-L` is set to non-control mitochondrial region, corresponding to position range 576-16024 bp. The variants in the control region of mitochondria are called from reads aligned to the shifted reference and the interval is set to 8025-9144 bp, which corresponds to the location of the control region in the shifted reference genome.

4.2.4 Merging variant calls

The merging module encapsulates the functionality required to merge variant calls from control and non-control regions. The variants called in the control region (using shifted mitochondrial reference) are transformed back to coordinates of canonical reference sequence using Picard LiftoverVcf tool:

```
gatk LiftoverVcf -I <VCF> -O <OUT_VCF>
    -R <REFERENCE>
    --CHAIN <SHIFT_BACK_CHAIN>
    --REJECT <REJECTED_VCF>
```

To liftover the VCF file back to canonical reference, the LiftoverVcf tool requires a chain file (`--CHAIN` option), alongside the reference and

input VCF file. As described in section 4.1, the chain files are generated by the process of shifting reference FASTA files and used as input for lifting over the shifted VCF file. The `--REJECT` option specifies a file to write rejected records, which should be empty in an ideal case.

The lifted-over VCF is then merged with VCF containing variants from the non-control region using Picard MergeVcfs tool:

```
gatk MergeVcfs -I <[VCFS]> -O <MERGED_VCF>
```

Additionally, the stats files of the control and non-control region variant files are merged using the following command:

```
gatk MergeMutectStats -stats <[STATS]> -O <MERGED_STATS>
```

4.2.5 Postprocessing

The postprocessing stage of raw variant calls includes normalization and application of multiple filters. The initial filtering phase includes filtering variants based on multiple specified parameters using GATK FilterMutectCalls in mitochondrial mode:

```
gatk FilterMutectCalls -V <VCF> -O <OUT_VCF>
    -R <REFERENCE> --stats <STATS>
    --max-alt-allele-count 4
    --mitochondria-mode
    --min-allele-fraction 0
    --f-score-beta 1
    --contamination-estimate 0.0
```

The `--max-alt-allele-count` specifies a threshold for filtering out variants with too many alternate alleles. To filter out variants with low allele fractions, the minimum threshold can be set through the `--min-allele-fraction` option. The `--f-score-beta` specifies a relative weight of recall and precision, setting it to a greater value than default 1.0 tweaks the results against greater recall, the lower values mean greater precision. Optionally, the variants are filtered based on estimated contamination by setting the `--contamination-estimate` parameter. To estimate the level of contamination in mitochondrial DNA samples, we utilize haplocheck [51]. The module provides an

option to tune all mentioned filtering parameters by the user, while the defaults are set based on the recommendations.

After applying initial filters, the variants are further filtered based on the list of common artifacts, also referred to as "*blacklist*". The default blacklists in BED format are provided for rCRS and RSRS references as a part of *mitopy* package. The rCRS blacklist was obtained from the GATK resource bundle. The RSRS was derived from the rCRS blacklist, containing the same sites plus additional sites containing spacers "*N*" (to preserve rCRS sequence numbering) at positions 523 and 524. User can specify a custom blacklist. To filter blacklisted sites, the GATK VariantFiltration tool is run with the following command:

```
gatk VariantFiltration -V <VCF> -O <OUT_VCF>
    --apply-allele-specific-filters
    --mask <BLACKLIST>
    --mask-name blacklisted_site
```

The specific blacklist is passed by the `--mask` option. The tool compares variants against the mask file and adds `--mask-name` to a FILTER field of the VCF file if the variant overlaps regions specified in the mask file.

The final filtering layer includes filtering against potential NuMTs. We use GATK NuMTFilterTool, which detects potential NuMTs based on median autosomal coverage (`--autosomal-coverage` option):

```
gatk NuMTFilterTool -V <VCF> -O <OUT_VCF>
    -R <REFERENCE>
    --autosomal-coverage <INT>
```

Following filtering, the individual variant calls are left-aligned, multiallelic sites are split and non-passing variants are removed using following commands:

```
gatk LeftAlignAndTrimVariants -V <VCF> -O <OUT_VCF>
    -R <REFERENCE>
    --split-multi-allelics
    --dont-trim-alleles
    --keep-original-ac

gatk SelectVariants -V <VCF> -O <OUT_VCF>
    --exclude-filtered
```

4.2.6 Calculating coverage

To obtain coverage per base for reads aligned to the mitochondrial region using a double alignment strategy, the coverage has to be computed for reads aligned to the non-control region (canonical reference) and control region (shifted reference) separately and concatenated in the final step. We compute coverage per-base from the BAM file containing reads aligned to the canonical reference, as well as the BAM file with reads aligned to shifted reference using the following command:

```
mosdepth <PREFIX> <BAM>
```

Mosdepth outputs coverage per base in BED file format. The BED file contains positions in a 0-based, half-open [start-1, end) coordinate system. To match positioning of the variant VCF files, which uses a 1-based coordinate system, we convert the positions from 0-based to 1-based coordinate system. The coverage per-base from the control and non-control region is then combined and written to a CSV file. Optionally, the interactive plot in HTML format is created using plotly [73].

4.2.7 Identifying haplogroups

The sample haplogroup is identified using Haplogrep3 with the following command:

```
haplogrep3 classify
    --in <VCF>
    --out <REPORT>
    --tree <TREE>
```

The phylogenetic tree that will be used for identifying the haplogroup is specified by the `--tree` option. For the rCRS reference genome, we use phylotree `phylotree-rcrs@17.2`, which is the latest version of phylotree for rCRS available by default. The latest version of the phylotree for the RSRS sequence has to be additionally installed with the following command:

```
haplogrep3 install-tree phylotree-rsrs@17.1
```

4.2.8 Annotation

We implemented a custom offline annotation module, which performs functional annotation using SnpEff and additionally integrates information from multiple sources and databases. It outputs an annotated VCF file and optionally exports the annotations to human-readable CSV format. The annotation module expects variants in the VCF file to be normalized. The annotation is performed with respect to rCRS reference.

Functional annotation

Performing offline functional annotation of human data usually involves downloading large annotation databases. However, we do not essentially need to work with whole human genome databases as our main focus is on the mitochondrial genome. For this purpose, we built a custom annotation database containing data exclusively on the human mitochondrial genome (rCRS reference) from GenBank (.gbk) annotation file and FASTA file containing protein sequences downloaded from NCBI. The custom database was built using the following command:

```
snpeff build -v -genbank NC_012920
```

In order for the custom database to be registered by SnpEff, the following lines specifying the database name, contained chromosomes, and special codon table were added to the SnpEff configuration file:

```
NC_012920.genome : NC_012920
    NC_012920.chromosomes : chrM
    NC_012920.chrM.codonTable : Vertebrate_Mitochondrial
```

The prebuilt database, together with the custom SnpEff configuration file are included in *mitopy* package.

The functional annotation is performed using the following command:

```
snpeff -config <CONFIG_FILE> -noStats
    -no-upstream -no-downstream
    <VCF> > <OUT_VCF>
```

We specify a path to our custom config file which includes a custom mitochondrial annotation database and disabled statistics together with upstream and downstream annotations to speed up the process. SnpEff adds annotations to the INFO field of the VCF file under the ANN tag. It includes variant effects based on Sequence Ontology terms (e.g., missense, synonymous, non-synonymous), estimation of impact on protein function, and further information on the affected genes and transcripts.

Additional annotations

For additional annotations, we gathered data from sources mentioned in section 3.2.8. The obtained annotation files had different file formats, such as VCF, plain TXT, or CSV/TSV. We refactored the annotation files to have the same file format, CSV. Additionally, when the annotation file contained data on the whole human genome, we extracted the data associated with the mitochondrial genome. We also unified the annotation files to include three columns named POS (position), REF (reference allele), and ALT (alternate allele), as these are used for matching the variant in the VCF file with annotation data. For the refactoring, we mainly used the pandas [79] Python library. The prepared annotation files are provided as part of the *mitopy* tool. We also include the refactoring scripts, as they can be leveraged for future updates of annotation files.

The annotation module adds additional annotations directly to INFO fields of the VCF file using VCFPy [4], a Python library providing an interface for reading and writing VCF files. Additionally, we adjust genotypes (GT entry in the FORMAT field of the VCF file) in the VCF file based on the minimum homoplasmy threshold. The homoplasmic variant is marked as 1/1 genotype, heteroplasmic variant corresponds to 0/1 genotype [23].

4. https://github.com/bihealth/vcfpy

Export to CSV

Optionally, annotations can be exported into human-readable CSV file format. The export of annotations from the VCF file to CSV is performed using SnpSift [80] tool using the following command:

```
snpsift extractFields -e . <VCF> <VCF_FIELDS>
```

We rename all columns, which include annotation fields from SnpEff adding "*SnpEff_*" prefix. We additionally include two genotype fields (from the FORMAT VCF fields) - allele frequency (AF) and Genotype (GT), as they indicate the type of mitochondrial variants with regard to their homoplasmy/heteroplasmy. We rename the AF field to "*Heteroplasmy Fraction*" and the GT field to "*MT Variant Type*". We replace genotypes 1/1 and 0/1 with their interpretations (homoplasmic/heteroplasmic).

4.2.9 Visualization

We create an interactive HTML visualization of variants from a VCF file using plotly [73]. The visualization consists of multiple layers. The base layer represents a circular mitochondrial genomic map with 4 types of genomic loci: CDS (protein-coding region), rRNA, tRNA, and regulatory region (D-loop) represented by different colors. By default, it depicts the regions together, optionally the regions can be split and visualized on corresponding strands where they belong to (H and L strand). On hover, concrete names of the genes are displayed. The mitochondrial genome annotation was obtained from the NCBI database.

Variants are visualized on a separate trace in the inner circle. We opted for the representation of variants at their respective positions in the form of scatterplot since scatterplot design allows for a clear representation of their spatial distribution across the mitochondrial genome. The distance of individual variants (points) from the inner trace border indicates their heteroplasmy fraction (the more distant from the bottom border, the higher the heteroplasmy fraction). On hover, basic information about a chosen variant is displayed, including its position, reference allele, alternate allele, and heteroplasmy fraction.

Optionally, the precalculated per-base coverage is included in the innermost circle of the plot, visualized through a line plot. The final plot can be optionally saved as a static image in PNG format.

Figure 4.1 shows an example visualization of variants identified in sample NA12878.

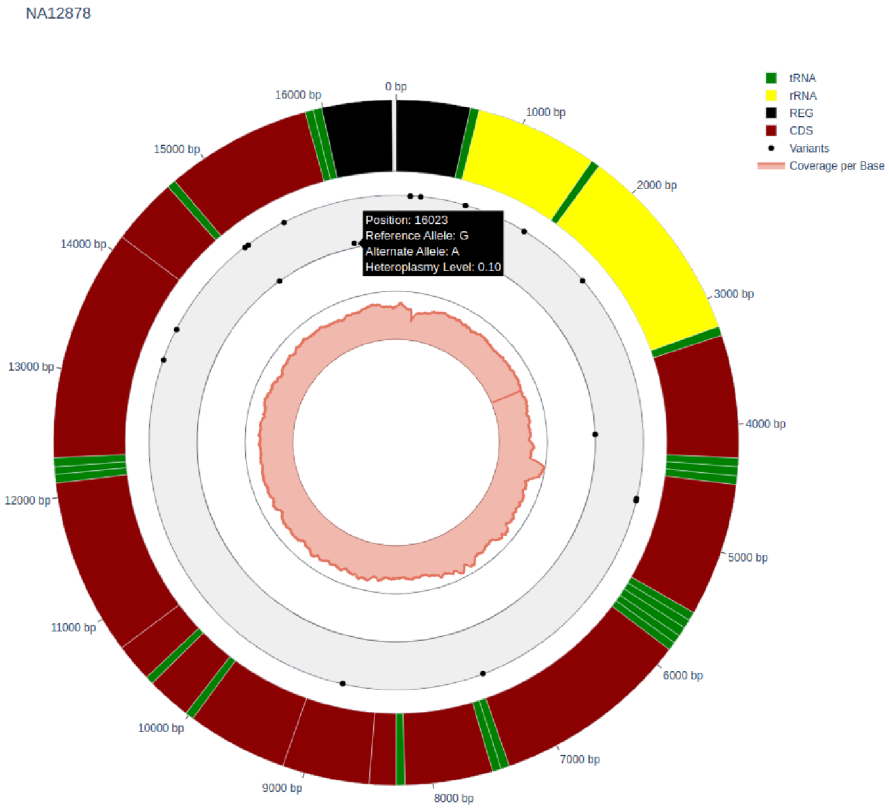


Figure 4.1: Visualization of variants. The outermost trace represents the mitochondrial genome map colored by the type of genomic locus. The variants trace shows variants with different heteroplasmic fractions. The innermost circle shows the coverage of the mitochondrial genome.

4.3 Docker

To satisfy the requirement of a simple installation process and platform independence, we set up an environment using Docker. We created a Docker image with all dependencies and external tools installed. Since the created tool is written in Python and depends on Python libraries, we use the official Python 3.11 slim image as the base image. In our tool, we also use external tools and toolkits which are Java-based and thus require a Java virtual machine to run. To satisfy the requirements of Java-based tools, we additionally install OpenJDK 17 inside the image. All external tools are installed using available pre-built versions.

Finally, to install our application and all its dependencies, we use a multi-stage Docker build process for optimization purposes. The Python dependencies of our application are managed by Poetry tool [81]. In the build stage, we use Python 3.11 buster image, which contains the necessary development dependencies required to install all Python dependencies in a virtual environment, but not required for runtime. The pre-built virtual environment with our application is then passed from the build stage to the runtime stage with a `COPY --from` instruction for multi-stage builds [82].

The `mitopy:latest` docker image was published on Dockerhub[5].

4.4 Automatization

We implemented DevOps principles, emphasizing the automation, continuous integration, and continuous delivery of our software. These principles were seamlessly integrated into a managed workflow environment using GitHub workflows. GitHub allowed us to define and automate steps triggered by changes, enabling continuous verification of our application. This approach ensures ongoing quality checks through automated testing and facilitates the consistent execution of Docker build and publish steps, maintaining consistency in our software delivery process.

5. `https://hub.docker.com/r/bendda/mitopy`

4.5 Nextflow implementation

To demonstrate the use of our application to build a scalable pipeline using a workflow manager of choice, which can be easily deployed to cloud-based environments or computing clusters, we implement our pipeline in Nextflow [83]. The choice of Nextflow as a workflow manager was based on its increasing popularity and one of the largest active user community [84], as well as the recently added support for Nextflow pipelines on the DNAnexus platform.

Nextflow is a workflow framework, which uses an extension of the Groovy programming language to simplify the process of writing complex computational workflows. The basic building block of the nextflow workflow is a process, an independent unit that executes a single script or command. The workflow consists of interconnected processes, which communicate together via so-called channels. Each process can declare channels as inputs and outputs, which then defines the pipeline execution flow itself [85].

To create a nextflow implementation based on our *mitopy* application, we wrap individual stages of the pipeline (*mitopy* commands) into separate processes and define input and main output channels. The main workflow script is contained in `main.nf` file, which imports individual processes from `modules.nf` file. The nextflow implementation provides an option to effectively run the pipeline on multiple samples simultaneously. The default configuration of the pipeline is defined in the `nextflow.config` file and includes declaration of pipeline parameters and configuration of individual processes. We define the location of the main outputs of the pipeline with `publishDir` directive. Nextflow engine provides a seamless integration with Docker containers, where processes are transparently executed. This feature also allows the pipelines to be executed on any platform which supports Docker technology. We enable Docker execution and specify a created image for our application, which will be used to run all processes, by adding the following lines to the nextflow configuration file:

```
process.container = 'bendda/mitopy:latest'
docker.enabled = true
```

Finally, we create the schema file (`nextflow_schema.json`) for the pipeline, which defines the constraints on workflow parameters. The nextflow implementation of the pipeline is available on GitHub [6].

4.6 Deployment on DNAnexus platform

DNAnexus is a cloud-based platform that provides tools and infrastructure for the management and analysis of genomic data. It enables researchers and organizations to store, analyze, and collaborate on large-scale DNA sequencing projects in a secure and scalable environment [86].

Recently, DNAnexus announced enhanced support for nextflow pipelines, allowing users to easily import and build their own nextflow pipelines among other features [87]. The nexflow pipeline can be imported from a remote repository or local disk space as a DNAnexus executable, i.e. app or applet. We import the pipeline from the remote repository via the command-line interface (DNAnexus Platform SDK) using the following command, where we specify the repository URL and destination project on the platform:

```
dx build --nextflow \
  --repository https://github.com/bendda/mitopy-nf \
  --destination project-xxxx:/applets/mitopy-nf
```

This imports the pipeline as an executable applet which can be easily run with specified parameters using command:

```
dx run project-xxxx:/applets/mitopy-nf \
  -ialignments="dx://project-xxxx:/inputs/*.{bam,bai}" \
  -ioutdir="dx://project-xxxx:/outputs/" \
  --brief -y
```

The nextflow pipeline executable initiates a tree of jobs, featuring a head job responsible for running the Nextflow executor. Additionally, there are numerous subjobs, each executing a single process. While the pipeline progresses, the head job consistently maintains a "*running*" state, overseeing the overall execution of the job tree [88].

6. `https://github.com/bendda/mitopy-nf`

5 Testing

To ensure that the created application and all its parts produce desirable results, we created a suite of unit tests, each testing an individual module. Furthermore, we conducted end-to-end testing of the nextflow adaptation of the pipeline on real WGS data, assessing its performance, runtime, and resource utilization on both the local machine and the DNAnexus platform.

5.1 Unit testing

We created a suite of unit tests for each module of the implemented *mitopy* package. The main purpose of these tests is to validate the correctness of individual modules and steps, as well as identify potential issues through the development process. Unit tests play also a valuable role in the later maintenance and potential updates of the package. We created unit tests using pytest [89], a Python testing framework.

5.2 End-to-end testing

5.2.1 Running locally

For local testing, we selected 3 samples from phase 3 of 1000 Genomes Project [90] (HG00096, HG00114, HG00150). The low coverage alignment BAM files (each $\sim$ 13 GB) and respective index files were obtained from IGSR [91].

Using the nextflow-generated report, we were able to analyze the utilization of computing resources for each stage of the pipeline. In terms of CPU utilization, the postprocessing stage stands out due to its substantial computational requirements (see Figure 5.1). Variant calling within the non-control mitochondrial region (`CALL_MT_DEFAULT` process) is the most memory intensive, as well as time-consuming stage of the pipeline (see Figures 5.2, 5.3).

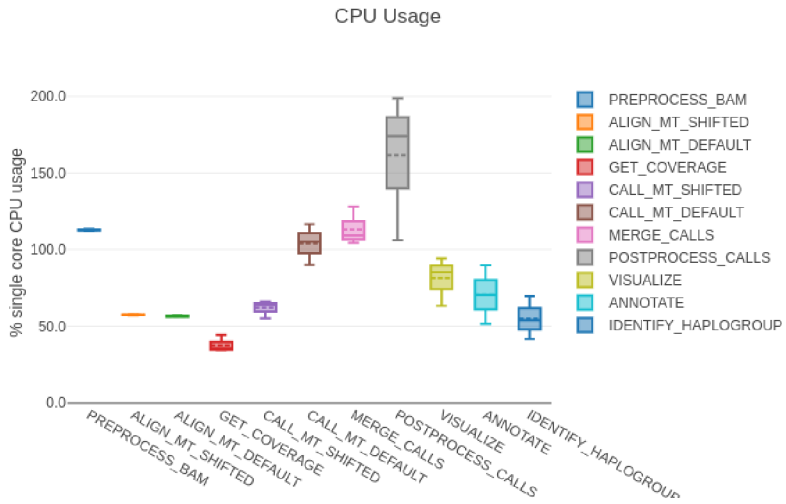


Figure 5.1: CPU allocation.

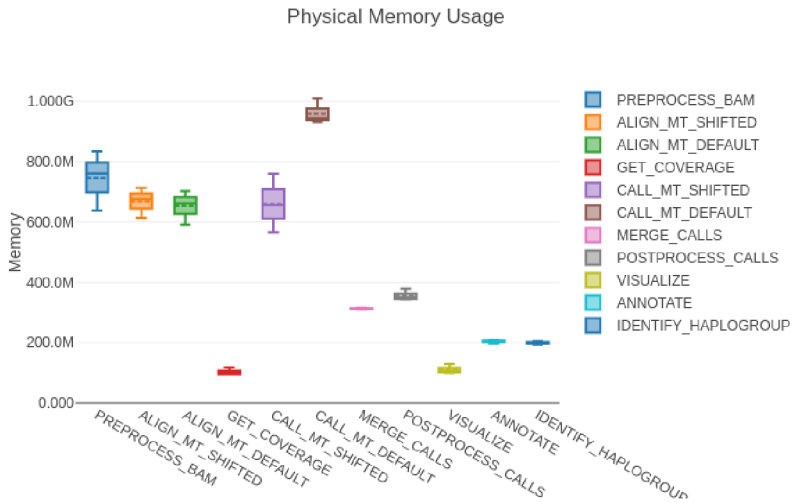


Figure 5.2: Memory utilization.

Figure 5.3: Execution time.

5.2.2 Running on DNAnexus

We performed a test run on the cloud-based DNAnexus platform using a medium-sized WGS alignment file ($\sim$ 13 GB). The test file was obtained from the GATK test data resource and contains WGS alignment for the NA12878 sample, which was generated as a part of Genome in a Bottle project [92] and downsampled to a size of 24 readgroups for testing purposes.

Figure 5.4 depicts the monitor tab in the user interface of the DNAnexus platform. It shows an execution tree of the pipeline, with the headjob and its subjobs, each corresponding to one stage of the pipeline. From the monitor tab, one can clearly see which stages were run in parallel. The mem2_ssd1_v2_x4 AWS instance type was used for the headjob as the default instance type, with 4 CPUs, 16 GB memory, and 150 GB disk space. The instance type of each subjob was dynamically determined based on the subjob's resource requirements [88]. The total runtime of the pipeline was 43 minutes. For the headjob, as well as all subjobs, an execution log can be viewed (see Figure 5.5).

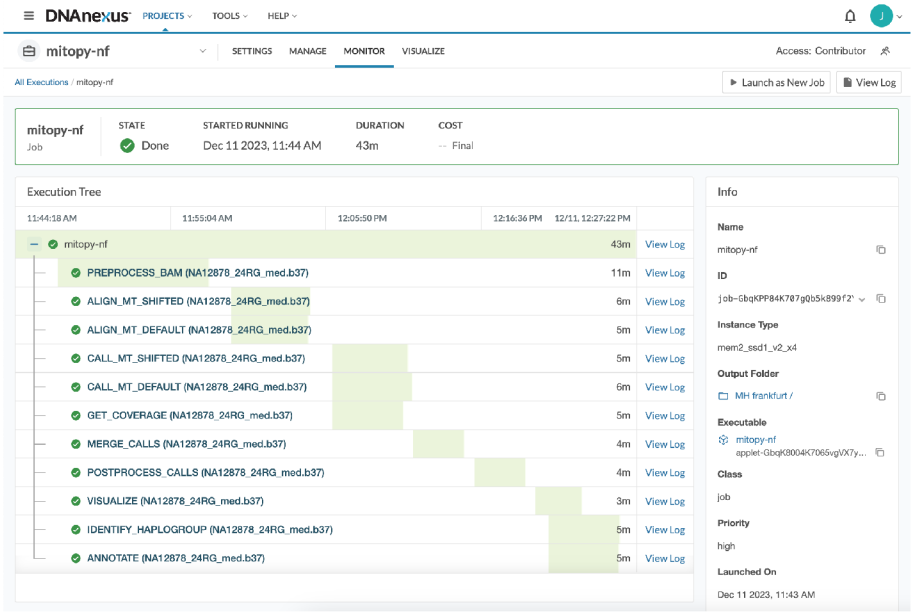


Figure 5.4: DNAnexus test run.

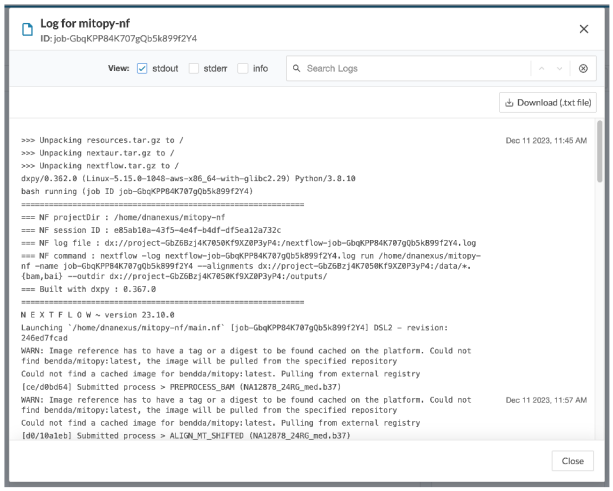


Figure 5.5: Part of the log of the headjob on the DNAnexus platform.

Conclusion

The main goal of this thesis was to understand key concepts of mitochondrial DNA variant detection and analysis. Based on the analysis, the aim was to create a comprehensive application capable of performing variant detection built upon best practices outlined by GATK and further enhance it by including functionality necessary for variant interpretation in a biological context.

The created application consists of a variant identification phase and a variant analysis phase. The variant identification phase is based on GATK best practices guidelines and offers an easy-to-use high-level interface, allowing users to tune only relevant set of parameters while hiding the default settings of underlying tools from the GATK suite. This can be beneficial for users not familiar with the GATK ecosystem, allowing them to apply GATK best practices to their research. The variant analysis phase provides functionality for extensive annotation, haplogroup identification, coverage calculation and visualization of identified mitochondrial variant calls. The application is supported by extensive documentation to further simplify its usage for end users.

Efficient management of the dependencies, portability, and easy installation process was achieved through container-based virtualization, concretely Docker.

The application has a modular design, providing an interface for running individual pipeline stages, as well as the end-to-end pipeline. The modular design provides flexibility, allowing users to integrate parts of the pipeline in their own pipelines or effectively build scalable pipelines in a workflow manager of choice, using our application as the underlying library. To demonstrate this capability, we created a nextflow implementation, which can run the pipeline efficiently on a large number of samples and can be easily deployed on cloud-based environments or computing clusters. The nextflow implementation of our pipeline was successfully deployed and run on the DNAnexus platform.

There exist several possible directions for future enhancements of the application. The variant analysis phase could be enhanced by integrating additional annotation sources and databases. Furthermore, the visualization can be further extended to include more valuable

information, such as summary statistics based on annotations etc. There is also room for improvement in terms of performance and resource utilization.

Bibliography

1. TUPPEN, Helen A.L.; BLAKELY, Emma L.; TURNBULL, Douglass M.; TAYLOR, Robert W. Mitochondrial DNA mutations and human disease. *Biochimica et Biophysica Acta (BBA) - Bioenergetics*. 2010, vol. 1797, no. 2, pp. 113–128. ISSN 0005-2728. Available from DOI: 10.1016/j.bbabio.2009.09.005.
2. MCKENNA, Aaron; HANNA, Matthew; BANKS, Eric; SIVACHENKO, Andrey; CIBULSKIS, Kristian; KERNYTSKY, Andrew; GARIMELLA, Kiran; ALTSHULER, David; GABRIEL, Stacey; DALY, Mark; DEPRISTO, Mark. The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *Genome research*. 2010, vol. 20, pp. 1297–303. Available from DOI: 10.1101/gr.107524.110.
3. VOSS, Kate; AUWERA, Geraldine A. Van der; GENTRY, Jeff. Full-stack genomics pipelining with GATK4 + WDL + Cromwell. *F1000Research*. 2017, vol. 6. Available from DOI: 10.7490/F1000RESEARCH.1114631.1.
4. HABBANE, Mouna; MONTOYA, Julio; RHOUDA, Taha; SBAOUI, Yousra; RADALLAH, Driss; EMPERADOR, Sonia. Human Mitochondrial DNA: Particularities and Diseases. *Biomedicines*. 2021, vol. 9. Available from DOI: 10.3390/biomedicines9101364.
5. SAGAN, Lynn. On the origin of mitosing cells. *Journal of Theoretical Biology*. 1967, vol. 14, no. 3, 225–IN6. ISSN 0022-5193. Available from DOI: 10.1016/0022-5193(67)90079-3.
6. GEIGER, Otto; SANCHEZ-FLORES, Alejandro; PADILLA-GOMEZ, Jonathan; ESPOSTI, Mauro Degli. Multiple approaches of cellular metabolism define the bacterial ancestry of mitochondria. *Science Advances*. 2023, vol. 9, no. 32, eadh0066. Available from DOI: 10.1126/sciadv.adh0066.
7. ROGER, Andrew J.; MUÑOZ-GÓMEZ, Sergio A.; KAMIKAWA, Ryoma. The Origin and Diversification of Mitochondria. *Current Biology*. 2017, vol. 27, no. 21, R1177–R1192. ISSN 0960-9822. Available from DOI: 10.1016/j.cub.2017.09.015.

8. CARVALHO, Daniel; ANDRADE, Roberto; PINHO, Suani; GÓES-NETO, Aristóteles; PETIT LOBAO, Thierry; BOMFIM, Gilberto; EL-HANI, Charbel. What are the Evolutionary Origins of Mitochondria? A Complex Network Approach. *PLoS ONE*. 2015, vol. 10. Available from DOI: 10.1371/journal.pone.0134988.

9. TAANMAN, Jan-Willem. The mitochondrial genome: structure, transcription, translation and replication. *Biochimica et Biophysica Acta (BBA) - Bioenergetics*. 1999, vol. 1410, no. 2, pp. 103–123. ISSN 0005-2728. Available from DOI: 10.1016/S0005-2728(98)00161-3.

10. SATO, Miyuki; SATO, Ken. Maternal inheritance of mitochondrial DNA by diverse mechanisms to eliminate paternal mitochondrial DNA. *Biochimica et Biophysica Acta (BBA) - Molecular Cell Research*. 2013, vol. 1833, no. 8, pp. 1979–1984. ISSN 0167-4889. Available from DOI: 10.1016/j.bbamcr.2013.03.010.

11. GASPARRE, G.; PORCELLI, A.M. *The Human Mitochondrial Genome: From Basic Biology to Disease*. Elsevier Science, 2020. ISBN 9780128226421. Available also from: https://books.google.sk/books?id=c1TODwAAQBAJ.

12. PARAKATSELAKI, Maria-Eleni; LADOUKAKIS, Emmanuel D. mtDNA heteroplasmy: origin, detection, significance, and evolutionary consequences. *Life*. 2021, vol. 11, no. 7, p. 633. Available from DOI: 10.3390/life11070633.

13. RENSCH, Thomas; VILLAR, Diego; HORVATH, Julie; ODOM, Duncan; FLICEK, Paul. Mitochondrial heteroplasmy in vertebrates using ChIP-sequencing data. *Genome Biology*. 2016, vol. 17. Available from DOI: 10.1186/s13059-016-0996-y.

14. XUE, Liying; MOREIRA, Jesse; SMITH, Karan; FETTERMAN, Jessica. The Mighty NUMT: Mitochondrial DNA Flexing Its Code in the Nuclear Genome. *Biomolecules*. 2023, vol. 13, p. 753. Available from DOI: 10.3390/biom13050753.

15. CHIAL H., Craig J. mtDNA and mitochondrial diseases. *Nature Education*. 2008, vol. 1, no. 1, p. 217.

16. MITCHELL, Sabrina L; GOODLOE, Robert; BROWN-GENTRY, Kristin; PENDERGRASS, Sarah A; MURDOCK, Deborah G; CRAWFORD, Dana C. Characterization of mitochondrial haplogroups in a large population-based sample from the United States. *Human genetics*. 2014, vol. 133, pp. 861–868. Available from DOI: 10.1007/s00439-014-1421-9.

17. OVEN, Mannis van; KAYSER, Manfred. Updated comprehensive phylogenetic tree of global human mitochondrial DNA variation. *Human Mutation*. 2009, vol. 30, no. 2, E386–E394. Available from DOI: 10.1002/humu.20921.

18. ROECK, Alexander; DÜR, Arne; OVEN, Mannis van; PARSON, Walther. Concept for estimating mitochondrial DNA haplogroups using a maximum likelihood approach (EMMA). *Forensic science international. Genetics*. 2013, vol. 7. Available from DOI: 10.1016/j.fsigen.2013.07.005.

19. ZHU, Zhenhua; WANG, Xiangdong. Significance of Mitochondria DNA Mutations in Diseases. In: *Mitochondrial DNA and Diseases*. Ed. by SUN, Hongzhi; WANG, Xiangdong. Singapore: Springer Singapore, 2017, pp. 219–230. ISBN 978-981-10-6674-0. Available from DOI: 10.1007/978-981-10-6674-0_15.

20. LAX, Nichola; TURNBULL, Doug; REEVE, Amy. Mitochondrial Mutations: Newly Discovered Players in Neuronal Degeneration. *The Neuroscientist : a review journal bringing neurobiology, neurology and psychiatry*. 2011, vol. 17, pp. 645–58. Available from DOI: 10.1177/1073858411385469.

21. SPENCER, David H.; ZHANG, Bin; PFEIFER, John. Chapter 8 - Single Nucleotide Variant Detection Using Next Generation Sequencing. In: KULKARNI, Shashikant; PFEIFER, John (eds.). *Clinical Genomics*. Boston: Academic Press, 2015, pp. 109–127. ISBN 978-0-12-404748-8. Available from DOI: 10.1016/B978-0-12-404748-8.00008-3.

22. SEHN, Jennifer K. Chapter 9 - Insertions and Deletions (Indels). In: KULKARNI, Shashikant; PFEIFER, John (eds.). *Clinical Genomics*. Boston: Academic Press, 2015, pp. 129–150. ISBN 978-0-12-404748-8. Available from DOI: 10.1016/B978-0-12-404748-8.00009-5.

23. LARICCHIA, Kristen; LAKE, Nicole; WATTS, Nicholas; SHAND, Megan; HAESSLY, Andrea; GAUTHIER, Laura; BENJAMIN, David; BANKS, Eric; SOTO, Jose; GARIMELLA, Kiran; EMERY, James; REHM, Heidi; MACARTHUR, Daniel; TIAO, Grace; LEK, Monkol; MOOTHA, Vamsi; CALVO, Sarah. Mitochondrial DNA variation across 56,434 individuals in gnomAD. *Genome Research*. 2022, vol. 32, gr.276013.121. Available from DOI: 10.1101/gr.276013.121.

24. IP, Eddie; TROUP, Michael; XU, Colin; WINLAW, David; DUNWOODIE, Sally; GIANNOULATOU, Eleni. Benchmarking the Effectiveness and Accuracy of Multiple Mitochondrial DNA Variant Callers: Practical Implications for Clinical Application. *Frontiers in Genetics*. 2022, vol. 13, p. 692257. Available from DOI: 10.3389/fgene.2022.692257.

25. *About the GATK Best Practices* [online]. Broad Institute, 2023 [visited on 2023-01-11]. Available from: https://gatk.broadinstitute.org/hc/en-us/articles/360035894711-About-the-GATK-Best-Practices.

26. *Mitochondrial short variant discovery: SNVs & Indels* [online]. Broad Institute, 2023 [visited on 2023-01-11]. Available from: https://gatk.broadinstitute.org/hc/en-us/articles/4403870837275-Mitochondrial-short-variant-discovery-SNVs-Indels-.

27. *Next-generation sequencing* [online]. Genomics Education Programme, 2023 [visited on 2023-11-14]. Available from: https://www.genomicseducation.hee.nhs.uk/genotes/knowledge-hub/next-generation-sequencing/.

28. *A high-resolution view of the entire genome* [online]. Illumina, 2023 [visited on 2023-11-14]. Available from: https://www.illumina.com/techniques/sequencing/dna-sequencing/whole-genome-sequencing.html.

29. *Data pre-processing for variant discovery* [online]. Broad Institute, 2023 [visited on 2023-01-11]. Available from: https://gatk.broadinstitute.org/hc/en-us/articles/360035535912-Data-pre-processing-for-variant-discovery.

30. SHARMA, Himani; SINGH, Archna; SHARMA, Chandresh; JAIN, Sunesh; SINGH, Neeta. Mutations in the mitochondrial DNA D-loop region are frequent in cervical cancer. *Cancer cell international*. 2005, vol. 5, p. 34. Available from DOI: 10.1186/1475-2867-5-34.

31. ANDERSON, Sharon; BANKIER, Alan T; BARRELL, Bart G; BRUIJN, Maarten HL de; COULSON, Alan R; DROUIN, Jacques; EPERON, Ian C; NIERLICH, Donald P; ROE, Bruce A; SANGER, Frederick, et al. Sequence and organization of the human mitochondrial genome. *Nature*. 1981, vol. 290, no. 5806, pp. 457–465. Available from DOI: 10.1038/290457a0.

32. ANDREWS, Richard M; KUBACKA, Iwona; CHINNERY, Patrick F; LIGHTOWLERS, Robert N; TURNBULL, Douglass M; HOWELL, Neil. Reanalysis and revision of the Cambridge reference sequence for human mitochondrial DNA. *Nature genetics*. 1999, vol. 23, no. 2, pp. 147–147. Available from DOI: 10.1038/13779.

33. BEHAR, Doron; OVEN, Mannis van; ROSSET, Saharon; METSPALU, Mait; LOOGVÄLI, Eva-Liis; SILVA, Nuno; KIVISILD, Toomas; TORRONI, Antonio; VILLEMS, Richard. A "Copernican" Reassessment of the Human Mitochondrial DNA Tree from Its Root (vol 90, pg 675, 2012). *American journal of human genetics*. 2012, vol. 90, pp. 675–84. Available from DOI: 10.1016/j.ajhg.2012.03.002.

34. BANDELT, Hans-Jürgen; KLOSS-BRANDSTÄTTER, Anita; RICHARDS, Martin; YAO, Yong-Gang; LOGAN, Ian. The case for the continuing use of the revised Cambridge Reference Sequence (rCRS) and the standardization of notation in human mitochondrial DNA studies. *Journal of human genetics*. 2013, vol. 59. Available from DOI: 10.1038/jhg.2013.120.

35. INGMAN, Max; KAESSMANN, Henrik; PÄÄBO, Svante; GYLLENSTEN, Ulf. correction: Mitochondrial genome variation and the origin of modern humans. *Nature*. 2001, vol. 410. Available from DOI: 10.1038/35069127.

36. SHEN, Lishuang; ATTIMONELLI, Marcella; BAI, Renkui; LOTT, Marie; WALLACE, Douglas; FALK, Marni; GAI, Xiaowu. MSeqDR mvTool: A mitochondrial DNA Web and API resource for

comprehensive variant annotation, universal nomenclature collation, and reference genome conversion. *Human mutation*. 2018, vol. 39. Available from DOI: 10.1002/humu.23422.

37. MUZZEY, Dale; EVANS, Eric; LIEBER, Caroline. Understanding the Basics of NGS: From Mechanism to Variant Calling. *Current Genetic Medicine Reports*. 2015, vol. 3. Available from DOI: 10.1007/s40142-015-0076-8.

38. DAVIS, Ryan; KUMAR, Kishore; PUTTICK, Clare; LIANG, Christina; AHMAD, Kate; EDEMA-HILDEBRAND, Fabienne; PARK, Jin; MINOCHE, Andre; GAYEVSKIY, Velimir; MALLAWAARACHCHI, Amali; CHRISTODOULOU, John; SCHOFIELD, Deborah; DINGER, Marcel; COWLEY, Mark; SUE, Carolyn. Use of Whole-Genome Sequencing for Mitochondrial Disease Diagnosis. *Neurology*. 2022, vol. 99, 10.1212/WNL.0000000000200745. Available from DOI: 10.1212/WNL.0000000000200745.

39. KOBOLDT, Daniel. Best practices for variant calling in clinical sequencing. *Genome Medicine*. 2020, vol. 12. Available from DOI: 10.1186/s13073-020-00791-w.

40. TAN, Adrian; ABECASIS, Gonçalo R.; KANG, Hyun Min. Unified representation of genetic variants. *Bioinformatics*. 2015, vol. 31, no. 13, pp. 2202–2204. ISSN 1367-4803. Available from DOI: 10.1093/bioinformatics/btv112.

41. MCCARTHY, Davis; HUMBURG, Peter; KANAPIN, Alexander; RIVAS, Manuel; GAULTON, Kyle; CAZIER, Jean-Baptiste; DONNELLY, Peter. Choice of transcripts and software has a large effect on variant annotation. *Genome medicine*. 2014, vol. 6, p. 26. Available from DOI: 10.1186/gm543.

42. AKALIN, Altuna. *Computational Genomics with R*. 1st. Chapman and Hall/CRC, 2020. Available from DOI: 10.1201/9780429084317. Original work published 2020.

43. LI, Heng; HANDSAKER, Robert E.; WYSOKER, Alec; FENNELL, Tim; RUAN, Jue; HOMER, Nils; MARTH, Gabor T.; ABECASIS, Gonçalo R.; DURBIN, Richard. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*. 2009, vol. 25, pp. 2078–2079. Available from DOI: 10.1093/bioinformatics/btp352.

44. BONFIELD, James K. CRAM 3.1: advances in the CRAM file format. *Bioinformatics*. 2022, vol. 38, no. 6, pp. 1497–1503. ISSN 1367-4803. Available from DOI: 10.1093/bioinformatics/btac010.

45. *uBAM - Unmapped BAM Format*. Broad Institute, 2023. Available also from: https://gatk.broadinstitute.org/hc/en-us/articles/360035532132-uBAM-Unmapped-BAM-Format.

46. P, Danecek; A, Auton; ABECASIS, Goncalo; CA, Albers; BANKS, Emily; MA, DePristo; HANDSAKER, Robert; G, Lunter; SHERRY, Stephen; G, McVean; R, Genomes; D, Altshuler; D, Bentley; A, Chakravarti; A, Clark; F, De; P, Donnelly; Z, Ren. The variant call format and VCFtools. *Bioinformatics*. 2011, vol. 27, pp. 2156–8. Available from DOI: 10.1093/bioinformatics/btr330.

47. *Picard* [online]. Broad Institute, 2023 [visited on 2023-11-01]. Available from: http://broadinstitute.github.io/picard/.

48. LI, Heng. Fast and Accurate Short Read Alignment with Burrows-Wheeler Transform. *Bioinformatics (Oxford, England)*. 2009, vol. 25, pp. 1754–60. Available from DOI: 10.1093/bioinformatics/btp324.

49. VASIMUDDIN, Md.; MISRA, Sanchit; LI, Heng; ALURU, Srinivas. Efficient Architecture-Aware Acceleration of BWA-MEM for Multicore Systems. In: *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2019, pp. 314–324. Available from DOI: 10.1109/IPDPS.2019.00041.

50. BENJAMIN, David; SATO, Takutop; LICHTENSTEIN, Lee; STEWART, Chip; GETZ, Gad; CIBULSKIS, Kristian. Calling Somatic SNVs and Indels with Mutect2. 2019. Available from DOI: 10.1101/861054.

51. WEISSENSTEINER, Hansi; FORER, Lukas; FENDT, Liane; KHEIRKHAH, Azin; SALAS, Antonio; KRONENBERG, Florian; SCHOENHERR, Sebastian. Contamination detection in sequencing studies using the mitochondrial phylogeny. *Genome Research*. 2021, vol. 31. Available from DOI: 10.1101/gr.256545.119.

52. QUINLAN, Aaron R; HALL, Ira M. BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*. 2010, vol. 26, no. 6, pp. 841–842. Available from DOI: 10.1093/bioinformatics/btq033.

53. TARASOV, Artem; VILELLA, Albert J; CUPPEN, Edwin; NIJMAN, Isaac J; PRINS, Pjotr. Sambamba: fast processing of NGS alignment formats. *Bioinformatics*. 2015, vol. 31, no. 12, pp. 2032–2034. Available from DOI: 10.1093/bioinformatics/btv098.

54. PEDERSEN, Brent; QUINLAN, Aaron. mosdepth: quick coverage calculation for genomes and exomes. 2017. Available from DOI: 10.1101/185843.

55. KLOSS-BRANDSTÄTTER, Anita; PACHER, Dominic; SCHOENHERR, Sebastian; WEISSENSTEINER, Hansi; BINNA, Robert; SPECHT, Guenther; KRONENBERG, Florian. HaploGrep: A Fast and Reliable Algorithm for Automatic Classification of Mitochondrial DNA Haplogroups. *Human Mutation*. 2011, vol. 32, pp. 25–32. Available from DOI: 10.1002/humu.21382.

56. JAGADEESAN, Anuradha; EBENESERSDÓTTIR, S; GUÐMUNDSDÓTTIR, Valdis; THORDARDOTTIR, Elisabet; MOORE, Kristjan; HELGASON, Agnar. HaploGrouper: A generalized approach to haplogroup classification. *Bioinformatics (Oxford, England)*. 2020, vol. 37. Available from DOI: 10.1093/bioinformatics/btaa729.

57. OLIVARES, Víctor; MUÑOZ BARRERA, Adrián; LORENZO SALAZAR, José; ZARAGOZA-TRELLO, Carlos; RUBIO-RODRÍGUEZ, Luis; USERA, Ana; JÁSPEZ, David; IÑIGO-CAMPOS, Antonio; GONZÁLEZ-MONTELONGO, Rafaela; FLORES, Carlos. A benchmarking of human mitochondrial DNA haplogroup classifiers from whole-genome and whole-exome sequence data. *Scientific Reports*. 2021, vol. 11. Available from DOI: 10.1038/s41598-021-99895-5.

58. WEISSENSTEINER, Hansi; PACHER, Dominic; KLOSS-BRANDSTÄTTER, Anita; FORER, Lukas; SPECHT, Guenther; BANDELT, Hans-Jürgen; KRONENBERG, Florian; SALAS, Antonio; SCHOENHERR, Sebastian. HaploGrep 2: mitochondrial

haplogroup classification in the era of high-throughput sequencing. *Nucleic Acids Research*. 2016, vol. 44, gkw233. Available from DOI: 10.1093/nar/gkw233.

59. SCHOENHERR, Sebastian; WEISSENSTEINER, Hansi; KRONENBERG, Florian; FORER, Lukas. Haplogrep 3 - an interactive haplogroup classification and analysis platform. *Nucleic acids research*. 2023, vol. 51. Available from DOI: 10.1093/nar/gkad284.

60. WANG, Kai; LI, Mingyao; HAKONARSON, Hakon. ANNOVAR: functional annotation of genetic variants from high-throughput sequencing data. *Nucleic Acids Research*. 2010, vol. 38, no. 16, e164–e164. ISSN 0305-1048. Available from DOI: 10.1093/nar/gkq603.

61. MCLAREN, William; GIL, Laurent; HUNT, Sarah; RIAT, Harpreet; RITCHIE, Graham; THORMANN, Anja; FLICEK, Paul; CUNNINGHAM, Fiona. The Ensembl Variant Effect Predictor. *Genome Biology*. 2016, vol. 17. Available from DOI: 10.1186/s13059-016-0974-4.

62. CINGOLANI, Pablo; PLATTS, Adrian; WANG, Le; COON, Melissa; NGUYEN, Tung; LUAN, Wang; LAND, Susan; LU, Xiangyi; RUDEN, Douglas. A program for annotating and predicting the effects of single nucleotide polymorphisms, SnpEff. *Fly*. 2012, vol. 6, pp. 80–92. Available from DOI: 10.4161/fly.19695.

63. GUDMUNDSSON, Sanna; SINGER-BERK, Moriel; WATTS, Nicholas A; PHU, William; GOODRICH, Julia K; SOLOMONSON, Matthew; CONSORTIUM, Genome Aggregation Database; REHM, Heidi L; MACARTHUR, Daniel G; O'DONNELL-LURIA, Anne. Variant interpretation using population databases: Lessons from gnomAD. *Human mutation*. 2022, vol. 43, no. 8, pp. 1012–1030. Available from DOI: 10.1002/humu.24309.

64. CHEN, Siwei; FRANCIOLI, Laurent C.; GOODRICH, Julia K.; COLLINS, Ryan L.; KANAI, Masahiro; WANG, Qingbo; ALFÖLDI, Jessica; WATTS, Nicholas A.; VITTAL, Christopher; GAUTHIER, Laura D.; POTERBA, Timothy; WILSON, Michael W.; TARASOVA, Yekaterina; PHU, William; YOHANNES, Mary T.; KOENIG, Zan; FARJOUN, Yossi; BANKS, Eric; DONNELLY, Stacey; GABRIEL, Stacey; GUPTA, Namrata; FERRIERA, Steven; TOLONEN, Charlotte; NOVOD, Sam; BERGELSON, Louis;

ROAZEN, David; RUANO-RUBIO, Valentin; COVARRUBIAS, Miguel; LLANWARNE, Christopher; PETRILLO, Nikelle; WADE, Gordon; JEANDET, Thibault; MUNSHI, Ruchi; TIBBETTS, Kathleen; PROJECT CONSORTIUM, gnomAD; O'DONNELL-LURIA, Anne; SOLOMONSON, Matthew; SEED, Cotton; MARTIN, Alicia R.; TALKOWSKI, Michael E.; REHM, Heidi L.; DALY, Mark J.; TIAO, Grace; NEALE, Benjamin M.; MACARTHUR, Daniel G.; KARCZEWSKI, Konrad J. A genome-wide mutational constraint map quantified from variation in 76,156 human genomes. *bioRxiv*. 2022. Available from DOI: 10.1101/2022.03.20.485034.

65. SONNEY, Sanjay; LEIPZIG, Jeremy; LOTT, Marie; ZHANG, Shiping; PROCACCIO, Vincent; WALLACE, Douglas; SONDHEIMER, Neal. Predicting the pathogenicity of novel variants in mitochondrial tRNA with MitoTIP. *PLOS Computational Biology*. 2017, vol. 13, e1005867. Available from DOI: 10.1371/journal.pcbi.1005867.

66. NIROULA, Abhishek; VIHINEN, Mauno. PON-mt-tRNA: A multifactorial probability-based method for classification of mitochondrial tRNA variations. *Nucleic acids research*. 2016, vol. 44. Available from DOI: 10.1093/nar/gkw046.

67. VASER, Robert; ADUSUMALLI, Swarnaseetha; LENG, Sim; SIKIC, Mile; NG, Pauline. SIFT missense predictions for genomes. *Nature protocols*. 2015, vol. 11, pp. 1–9. Available from DOI: 10.1038/nprot.2015.123.

68. LOTT, Marie; LEIPZIG, Jeremy; DERBENEVA, Olga; XIE, Hongbo; CHALKIA, Dimitra; SARMADY, Mahdi; PROCACCIO, Vincent; WALLACE, Douglas. mtDNA Variation and Analysis Using Mitomap and Mitomaster. *Current protocols in bioinformatics*. 2013, vol. 44. Available from DOI: 10.1002/0471250953.bi0123s44.

69. LANDRUM, Melissa; LEE, Jennifer; BENSON, Mark; BROWN, Garth; CHAO, Chen; CHITIPIRALLA, Shanmuga; GU, Baoshan; HART, Jennifer; HOFFMAN, Douglas; JANG, Wonhee; KARAPETYAN, Karen; KATZ, Kenneth; LIU, Chunlei; MADDIPATLA, Zenith; MALHEIRO, Adriana; MCDANIEL, Kurt; OVETSKY, Michael; RILEY, George; ZHOU, George; MAGLOTT, Donna.

ClinVar: Improving access to variant interpretations and supporting evidence. *Nucleic acids research*. 2017, vol. 46. Available from DOI: 10.1093/nar/gkx1153.

70. POLLARD, Katherine; HUBISZ, Melissa; ROSENBLOOM, Kate; SIEPEL, Adam. Detection of non-neutral substitution rates on Mammalian phylogenies. *Genome research*. 2009, vol. 20, pp. 110–21. Available from DOI: 10.1101/gr.097857.109.

71. SIEPEL, Adam; BEJERANO, Gill; PEDERSEN, Jakob S; HINRICHS, Angie S; HOU, Minmei; ROSENBLOOM, Kate; CLAWSON, Hiram; SPIETH, John; HILLIER, LaDeana W; RICHARDS, Stephen, et al. Evolutionarily conserved elements in vertebrate, insect, worm, and yeast genomes. *Genome research*. 2005, vol. 15, no. 8, pp. 1034–1050. Available from DOI: 10.1101/gr.3715005.

72. KAROLCHIK, Donna; HINRICHS, Angela; FUREY, Terrence; ROSKIN, Krishna; SUGNET, Charles; HAUSSLER, David; KENT, W. The UCSC Table Browser data retrieval tool. *Nucleic acids research*. 2004, vol. 32, pp. D493–6. Available from DOI: 10.1093/nar/gkh103.

73. INC., Plotly Technologies. *Collaborative data science*. Montreal, QC: Plotly Technologies Inc., 2015. Available also from: https://plot.ly.

74. ANDERSON, Thomas; PETERSON, Larry; SHENKER, Scott; TURNER, Jonathan. Overcoming the Internet impasse through virtualization. *Computer*. 2005, vol. 38, pp. 34–41. Available from DOI: 10.1109/MC.2005.136.

75. EDER, Michael. Hypervisor- vs. Container-based Virtualization. 2016. Available from DOI: 10.2313/NET-2016-07-1_01.

76. HANUSSEK, Maximilian; BARTUSCH, Felix; KRÜGER, Jens. Performance and scaling behavior of bioinformatic applications in virtualization environments to create awareness for the efficient use of compute resources. *PLoS Computational Biology*. 2021, vol. 17. Available from DOI: 10.1371/journal.pcbi.1009244.

77. *Top Ten Programming Languages for Bioinformatics in 2023* [online]. Omics tutorials, 2023 [visited on 2023-12-07]. Available from: https://omicstutorials.com/top-ten-programming-languages-for-bioinformatics-in-2023/.

78. *Sphinx documentation* [online]. Spinx, 2023 [visited on 2023-11-30]. Available from: https://www.sphinx-doc.org/en/master/.

79. TEAM, The pandas development. *pandas-dev/pandas: Pandas*. Zenodo, 2020. Latest. Available from DOI: 10.5281/zenodo.3509134.

80. CINGOLANI, P.; PATEL, V.M.; COON, M.; NGUYEN, T.; LAND, S.J.; RUDEN, D.M.; LU, X. Using Drosophila melanogaster as a model for genotoxic chemical mutational studies with a new program, SnpSift. *Frontiers in Genetics*. 2012, vol. 3. Available from DOI: 10.3389/fgene.2012.00035.

81. EUSTACE, Sébastien; THE POETRY CONTRIBUTORS. *Poetry: Python packaging and dependency management made easy*. [N.d.]. Available also from: https://github.com/python-poetry/poetry.

82. *Multi-stage builds* [online]. Docker Docs, 2023 [visited on 2023-11-30]. Available from: https://docs.docker.com/build/building/multi-stage/.

83. DI TOMMASO, Paolo; CHATZOU, Maria; FLODEN, Evan W.; BARJA, Pablo; PALUMBO, Emilio; NOTREDAME, Cedric. Nextflow enables reproducible computational workflows. *Nature Biotechnology*. 2017, vol. 35, pp. 316–319. Available from DOI: 10.1038/nbt.3820.

84. WRATTEN, Laura; WILM, Andreas; GÖKE, Jonathan. Reproducible, scalable, and shareable analysis pipelines with bioinformatics workflow managers. *Nature methods*. 2021, vol. 18, no. 10, pp. 1161–1168. ISSN 1548-7091. Available from DOI: 10.1038/s41592-021-01254-9.

85. *Basic concepts* [online]. Nextflow, 2023 [visited on 2023-11-30]. Available from: https://www.nextflow.io/docs/latest/basic.html.

86. *DNAnexus Case Study* [online]. AWS, 2014 [visited on 2023-11-05]. Available from: https://aws.amazon.com/solutions/case-studies/dnanexus/.

87. *Announcing Enhanced Nextflow Support* [online]. DNAnexus, Year [visited on 2023-12-07]. Available from: https://blog.dnanexus.com/announcing-enhanced-nextflow-support.

88. *Running Nextflow Pipelines* [online]. DNAnexus Documentation, 2023 [visited on 2023-12-07]. Available from: https://documentation.dnanexus.com/user/running-apps-and-workflows/running-nextflow-pipelines.

89. KREKEL, Holger; OLIVEIRA, Bruno; PFANNSCHMIDT, Ronny; BRUYNOOGHE, Floris; LAUGHER, Brianna; BRUHIN, Florian. *pytest 7.4.0*. 2004. Available also from: https://github.com/pytest-dev/pytest.

90. A, Auton; ABECASIS, Goncalo; DM, Altshuler; RM, Durbin; DR, Bentley; A, Chakravarti; AG, Clark; P, Donnelly; EE, Eichler; P, Flicek; SB, Gabriel; GIBBS, Richard; ED, Green; ME, Hurles; KNOPPERS, Bartha; JO, Korbel; ES, Lander; LEE, Charles; LEHRACH, Hans; JA, Schloss. A global reference for human genetic variation. *Nature*. 2015, vol. 526, p. 68. Available from DOI: 10.1038/nature15393.

91. FAIRLEY, Susan; LOWY-GALLEGO, Ernesto; PERRY, Emily; FLICEK, Paul. The International Genome Sample Resource (IGSR) collection of open human genomic variation resources. *Nucleic Acids Research*. 2019, vol. 48, no. D1, pp. D941–D947. ISSN 0305-1048. Available from DOI: 10.1093/nar/gkz836.

92. ZOOK, Justin M; CATOE, David; MCDANIEL, Jennifer; VANG, Lindsay; SPIES, Noah; SIDOW, Arend; WENG, Ziming; LIU, Yuling; MASON, Christopher E; ALEXANDER, Noah, et al. Extensive sequencing of seven human genomes to characterize benchmark reference materials. *Scientific data*. 2016, vol. 3, no. 1, pp. 1–26. Available from DOI: 10.1101/026468.

A Source code

The source code can be found in the attached zip archive. It contains following folders:

- `mitopy`: source code for *mitopy* [1]
- `mitopy-nf`: source code for nextflow implementation of *mitopy* [2]

1. `https://github.com/bendda/mitopy`
2. `https://github.com/bendda/mitopy-nf`

B Manual

mitopy is a command-line toolkit to perform identification and analysis of short mitochondrial variants. The mitochondrial variant detection is based on GATK best practices. Functionality for analysis of detected variants includes annotation, visualization, haplogroup identification and coverage calculation. For detailed information, please consult documentation [1].

Prerequisities

The preferred way of running mitopy is through Docker. Please install Docker [2] and Docker Compose [3] on your system.

Run mitopy

Clone the repository:

```
git clone https://github.com/bendda/mitopy
```

Start the Docker container in interactive mode:

```
cd mitopy
docker compose run mitopy
```

Run the whole mitopy pipeline on example data:

```
mitopy run-pipeline \
    example_data/NA12878_20k_hg38.bam \
    -o example_data/outputs
```

Run tests

To run a suite of mitopy tests, run following command (in the cloned mitopy directory):

```
docker compose run test
```

1. https://mitopy.readthedocs.io
2. https://docs.docker.com/desktop/
3. https://docs.docker.com/compose/

Run nextflow implementation

To run the nextflow mitopy pipeline, please install Nextflow [4] and Docker [5] on your system.

To run the pipeline on example data, clone the repository:

```
git clone https://github.com/bendda/mitopy-nf.git
```

Run the nextflow mitopy pipeline on example data using following command:

```
cd mitopy-nf
nextflow run main.nf \
    --alignments "example_data/*.{bam,bai}" \
    --outdir results
```

4. https://www.nextflow.io/docs/latest/getstarted.html
5. https://docs.docker.com/desktop/

C Dockerfile

```
FROM python:3.11-buster as poetry-builder

RUN pip install poetry==1.4.2

ENV POETRY_NO_INTERACTION=1 \
    POETRY_VIRTUALENVS_IN_PROJECT=1 \
    POETRY_VIRTUALENVS_CREATE=1 \
    POETRY_CACHE_DIR=/tmp/poetry_cache

WORKDIR /app

COPY pyproject.toml poetry.lock ./
COPY . .

RUN --mount=type=cache,target=$POETRY_CACHE_DIR poetry install
     --no-root
RUN poetry run pip install .


FROM debian:bullseye-20211220-slim AS gatk4-build

ARG GATK_VERSION=4.4.0.0

ENV BUILD_PACKAGES unzip gcc
RUN apt-get update && apt-get install -y --no-install-
    recommends ${BUILD_PACKAGES}

WORKDIR /home
ADD https://github.com/broadinstitute/gatk/releases/download/${
    GATK_VERSION}/gatk-${GATK_VERSION}.zip gatk.zip
RUN unzip gatk.zip && \
    rm -rf gatk*/gatkdoc/ gatk*/gatk*spark.jar


FROM python:3.11-slim AS mitopy

ARG GATK_VERSION=4.4.0.0
```

```
ARG BWAMEM2_VERSION=2.2.1
ARG HAPLOCHECK_VERSION=1.3.3
ARG MOSDEPTH_VERSION=0.3.5
ARG HAPLOGREP3_VERSION=3.2.1

RUN mkdir /usr/local/bin/gatk4 /usr/local/bin/bwamem2

# Install OpenJDK-17
RUN apt-get update && \
    apt-get install -y openjdk-17-jre unzip lbzip2 && \
    apt-get clean;


ENV VIRTUAL_ENV=/app/.venv \
    PATH="/app/.venv/bin:$PATH"

COPY --from=poetry-builder ${VIRTUAL_ENV} ${VIRTUAL_ENV}

# haplocheck
ADD https://github.com/genepi/haplocheck/releases/download/v${
    HAPLOCHECK_VERSION}/haplocheck.zip haplocheck.zip
RUN unzip haplocheck.zip -d /usr/local/bin/haplocheck

# haplogrep3
ADD https://github.com/genepi/haplogrep3/releases/download/v${
    HAPLOGREP3_VERSION}/haplogrep3-${HAPLOGREP3_VERSION}-linux.
    zip haplogrep3.zip
RUN unzip haplogrep3.zip -d /usr/local/bin/haplogrep3

# mosdepth
ADD https://github.com/brentp/mosdepth/releases/download/v${
    MOSDEPTH_VERSION}/mosdepth /usr/local/bin/mosdepth
RUN chmod u+x /usr/local/bin/mosdepth

# bwamem2
ADD https://github.com/bwa-mem2/bwa-mem2/releases/download/v${
    BWAMEM2_VERSION}/bwa-mem2-${BWAMEM2_VERSION}_x64-linux.tar.
    bz2 bwa-mem2.tar.bz2
RUN tar -xf bwa-mem2.tar.bz2 -C /usr/local/bin/bwamem2 --strip-
    components=1 --no-same-owner
```

```
# snpEff
ADD https://snpeff.blob.core.windows.net/versions/
    snpEff_latest_core.zip snpeff.zip
RUN unzip snpeff.zip -d /usr/local/bin
RUN rm -rf /usr/local/bin/snpEff/examples/ /usr/local/bin/
    snpEff/galaxy/

RUN rm bwa-mem2.tar.bz2 haplocheck.zip snpeff.zip haplogrep3.
    zip

COPY --from=gatk4-build /home/gatk-${GATK_VERSION}/ /usr/local/
    bin/gatk4
COPY --from=gatk4-build /usr/lib/x86_64-linux-gnu/libgomp.so.1
     /usr/lib/x86_64-linux-gnu

ENV PATH="/usr/local/bin/haplogrep3:/usr/local/bin/snpEff/exec
    /:/usr/local/bin/mosdepth:/usr/local/bin/haplocheck:/usr/
    local/bin/bwamem2:/usr/local/bin/gatk4:${PATH}"

CMD [ "bash" ]
```