Introducing deep learning -based methods into the variant calling analysis pipeline

Master's Thesis

University of Turku

Department of Future Technologies

Master’s Degree Programme in Bioinformatics

2019

Vladislav Lysenkov

Abstract

Department of Future Technologies, University of Turku

VLADISLAV LYSENKOV: Introducing deep learning -based methods into the variant calling analysis pipeline

Supervisors: Sofia Khan, Laura Elo, Timo Knuutila

Master's Thesis, 64p

Master's Degree Programme in Bioinformatics

May 2019

Biological interpretation of the genetic variation enhances our understanding of normal and pathological phenotypes, and may lead to the development of new therapeutics.

However, it is heavily dependent on the genomic data analysis, which might be inaccurate due to the various sequencing errors and inconsistencies caused by these errors. Modern analysis pipelines already utilize heuristic and statistical techniques, but the rate of falsely identified mutations remains high and variable, particular sequencing technology, settings and variant type.

Recently, several tools based on deep neural networks have been published. The neural networks are supposed to find motifs in the data that were not previously seen.

The performance of these novel tools is assessed in terms of precision and recall, as well as computational efficiency. Following the established best practices in both variant detection and benchmarking, the discussed tools demonstrate accuracy metrics and computational efficiency that spur further discussion.

Keywords: next generation sequencing, variant calling, machine learning, deep learning, benchmark, accuracy, precision, recall

Acknowledgements

I would like to express my gratitude to my supervisors, Dr. Sofia Khan, Dr. Laura Elo and Prof. Timo Knuutila for their patience and timely guidance. In particular, I would like to thank Sofia for her eager engagement in scientific and otherwise discussion, Laura for sharing her strategic thinking, and Timo for challenging my understanding of the topics discussed.

I am thankful to my colleagues, especially Esko and Sami for their assistance with infrastructure; Markku for providing statistical advice; Satu for organizational support; Deepankar, Niklas and Ning for their insightful talk on related matters; and Maria, Mehrad, Thomas, Veronika and Xu for sharing their vision and friendly encouragement.

I wish to thank my family for their kindness and support during my period of study and their tolerance toward my choices and goals in life. Another warm thank you goes to my closest friends Michael and Victoria for keeping my spirits up.

Abbreviations

NGS / SGS: Next Generation Sequencing / Second Generation Sequencing

SBS: Sequencing By Synthesis

SBL: Sequencing By Ligation

CRT: Cyclic Reversible Termination

SMS: Single Molecule Sequencing

WGS: Whole Genome Sequencing

WES (WXS): Whole EXome Sequencing

DNA: Deoxyribonucleic Acid

RNA: Ribonucleic Acid

FFPE: Formalin-Fixed Paraffin-Embedded

SNP/SNV: Single Nucleotide Polymorphism / Single Nucleotide Variation

CIGAR: Concise Idiosyncratic Gapped Alignment Report

HMM: Hidden Markov Models

GMM: Gaussian Mixture Model

SGD: Stochastic Gradient Descent

ANN / DNN: Artificial Neural Network / Deep Neural Network

CNN / RNN: Convolutional Neural Network / Recurrent Neural Network

MLP: Multilayer Perceptron

TP: True Positive

FP: False Positive

TN: True Negative

FN: False Negative

Contents

Chapter 1: Introduction

This chapter introduces the objectives of the study, and covers the structure of this document.

Objectives

In the past several years, it has become apparent that the development and the adoption of next-generation sequencing technologies procured a massive amount of discoveries in molecular biology, genetics and medicine. Today, next-generation sequencing (NGS), also referred to as second generation sequencing (SGS) and high-throughput sequencing (HTS), is an umbrella term representing a range of principles, applications, and technologies.

The most widely used technologies include whole genome sequencing (WGS), targeted sequencing (from targeted regions to whole exome sequencing - WES/WXS), RNA-Seq (a study of RNAs that in practice is based on DNA sequencing), and others. They are used for assembling and annotating genomes, for studying genetic evolution and interactions and variations, gene expression, for predicting protein structure.

All these methods, including the methods for detecting genetic variations (variant calling), are essentially analyzing sequences of nucleotides that are produced with specific NGS platforms and procedures. A genomic data analysis relies on the understanding of the goal of the particular analysis, the sources and features and flaws of the data, and the techniques and tools in processing the data. The processing approach must be able to accommodate, for instance, the variability of the real data in respect to the sample size, sample origin (fresh tissue or formalin-fixed paraffin-embedded tissue), and sequencing protocol (target enrichment vs. amplification based).

By convention, such an analysis is conducted in the form of a pipeline that passes through the necessary steps, such as quality control, alignment to the reference genome, variant calling, as well as potential downstream analysis. An appropriate pipeline would not only take into account the variability of the data, but also be based on the best practices in the field and be convenient in use, efficient and accurate.

The aim of this work is the evaluation of the novel methods for variant calling with regard to the suitability to being a part of the variant calling pipeline. The pipeline is intended to be used mainly on NGS data produced with Illumina instruments (as the most abundant in the field). The required variant types for this pipeline are single nucleotide variations and small (less than 50 base pairs) insertions and deletions. The existing pipeline is based on the well-known best practices and tools from the Broad Institute; beside their functionality and recognition, they also have well-known intricacies and downsides. The utilization of the more appropriate (according to the

evaluation criteria) methods for identification of genomic variation would elevate the value of the analysis and could, eventually, lead to a better understanding of genetic causes of traits and diseases. Therefore, this work is relevant to the field.

Structure

Chapter 1 introduces this work, its objective and structure.

Chapter 2 introduces the motivation and the principles of genomic analysis and the methods of variant calling.

Chapter 3 elaborates on the modern, established methods of variant calling, as well as the machine learning background that is necessary for understanding them.

Chapter 4 provides further background on deep learning and examines the neural networks that were designed for the task.

Chapter 5 details how this work was organized and how the variant calling tools should be evaluated.

Chapter 6 compares the neural network based variant calling tools with the established method.

Chapter 7 concludes this work by discussing its progress and the results.

Chapter 2: Background on the genetic variation

This chapter describes modern genomic analysis. It starts with the very concept of the genetic variation, that is followed by a description of the methods for obtaining and analyzing such information. Both molecular mechanisms and computational processes are introduced, including challenges and applications. The focal point for the rest of the study is given.

Genetic variation

Genetic information

The genetic information, stored in DNA molecules in the form of nucleotide 'bases' — Adenine (A), Thymine (T), Guanine (G), and Cytosine (C) — affects the phenotype to the extent that of regulating functional development, its diversity, heritability and predisposition. The bases form sense and antisense strands of DNA molecules, that are complementary to each other, and therefore each base has a 'pair'. In literature, the term 'base pair', abbreviated as 'bp', amply denotes a unit of length of a nucleic acid sequence. The human genome (DNA molecules in the nucleus and in mitochondria) contains instructions that guide the development and interaction of the cells. Approximately 2% of the roughly 2994.26 bp long[1] human genome is transcribed to the protein-coding messenger RNAs (mRNAs) that get translated to the proteins and hence these genomic regions are called coding regions. The rest (more than 98%) consists of non-coding, regulatory sequences, instructing the production of RNAs that are involved in catalytic and maintenance processes[2]. Altogether, the coding and non-coding DNA provide the genetic instructions that are responsible for gene expression — the process of transcribing the DNA into the into RNA sequences and potential translation into the protein sequences[3].

The genetic code contains mutations, occurring due to the inner processes of the cell, such as damage repair or reproduction, or external mutagenic mechanisms, such exposure to harmful substances and ionizing radiation, interfering with nucleotides or bonds. These mutations, being either gain-of-function, equally viable or loss-of function changes, result in corresponding gene products. Thus, the changes concern various functional consequences, from the growth and movement and interplay of macromolecules to that of the cells.

Classification of genetic variation

Variants differ by their size, complexity, and frequency. One of the most basic, and common types of genetic variants is a point mutation (single nucleotide variation, or SNV), where a one nucleotide is substituted by another. For example, a sickle cell disease originates from the wrong amino acid (E/Glu instead of V/Val), translated from the corresponding coding region, where one nucleotide was replaced by another:

```
Wild-type:       ATGGTGCACCTGACTCCTGAGGAGAAGTCTGCCGTTACT...

Mutant:          ATGGTGCACCTGACTCCTGTGGAGAAGTCTGCCGTTACT...
```

Both of these sequences are parts of the β-globin protein gene. The first, wild-type sequence functions normally, producing proteins that form torus-like erythrocytes capable of carrying oxygen. The second, mutant (rs334 variant in the dbSNP variant database[4]) sequence causes erythrocytes to take a form that resembles a sickle, lowering their transporting ability[5].

Another frequently occurring type of small variants is an indel: an insertion or a deletion. Unlike SNV, an indel can span across more than one nucleotide, potentially up to 50 base pairs (bp) according to modern evaluation guidelines and variant truth sets[6]. In a case that an indel's length is not divisible by 3, i.e. it cannot be described as a composition of a whole number of codons (3-base sequences translated as amino acids), such an indel represents a frameshift mutation. With the reading frame being disrupted, the mRNA starts off the wrong nucleotide and thus produces a different set of amino acids (resulting in, most likely, an abnormal protein[7]). Non-frameshifting indels on the coding region, on the other hand, cause entire amino acids to be inserted into the amino acid chain or deleted from it.

In addition, insertions and deletions can be categorized as conservative and non-conservative depending on the place of their occurrence (between codons or inside a codon) and the subsequent effect on the neighboring amino acids. The majority of them are conservative; as studies show, indels that are or based on repetitive sequences (interspersed and tandem repeats) are abundant in the genome[8-11].

Tandem repeats are dynamic. In a case a repeat is located inside or near a gene, it's expansion may lead to severe neurodegenerative diseases. In the following well-studied example, a trinucleotide repeat expansion of the CAG sequence (Q/glutamine) significantly increases the rate (up to being a cause if the amount of repeats is 36 or more) of developing a Huntington disease[12-16]. The first sequence belongs to the reference genome GRCh38.p7, the second represents the rs71180116 variant from dbSNP:

```
TCCCTCAAGTCCTTC (CAG)19 CA ACAGCCGCCACCGCCGC

TCCCTCAAGTCCTTC (CAG)41 CA ACAGCCGCCACCGCCGC
```

The structural variants span beyond 50 bp and up to several megabase pairs (Mbp) and whole chromosomes (aneuploidy), typically being several kilobase pairs (kbp) long. This category is comprised of deletions, insertions, duplications, inversions, and translocations. It has been argued that their size and potential complexity, along with current sequencing methods, set limitations to their reliable discovery and characterisation[17-19]. As a result, experimental and computational approaches

focused on structural variants differ from the approaches focused on small variants. This work explores the latter.

Obtaining genetic information with NGS technologies

Modern sequencing of nucleic acids is, in essence, a parallel reading of millions of short fragments (hence the name - short-read massively parallel sequencing, or MPS) of the source nucleic acid. During the preparation of a library (a collection of reads), the DNA is extracted from the sample, physically or chemically fragmented, and its fragments are size-selected and amplified. Every individual fragment is composed of a sequence of nucleotides, and altogether they may constitute overlaps sufficient for further analysis with computational methods (sequence alignment, variant calling and interpretation). The likelihood of the correct determination of a particular nucleotide (base calling) depends on many factors, from the preparation of library to the exact method of determining the sequence[20].

Sample processing starts with the DNA extraction and purification[21-23]. For fresh and frozen samples, such as blood, buccal swab, bone marrow or other tissues, the nucleic acids are separated from the surroundings by the chemical, enzymatic or mechanical cell lysis that disrupts, degrades or breaks cellular membranes and components. For formalin-fixed paraffin-embedded (FFPE) tissues, common in clinical research, the paraffin must be dissolved first, and then chemically and enzymatically lysed. The detached DNA rests in lysate and must be purified using either phase extraction (centrifugation in a phenol-chloroform mixture) or column-based extraction (binding of nucleic acids to silica-based gel).

DNA fragmentation can be done via sonication (acoustic effect), hydroshearing (hydrodynamic effect), nebulization (atomizing effect), or enzymatic shearing (bond cleavage effect). The choice of the fragmentation method depends on the sample and fragmentation resources available as well as on the desired final fragment size, and may influence how the original sample is represented.

For instance, while the physical methods have higher reproducibility (more control over fragment sizes) and lower dependence on the actual sequence, they are not designed to scale and are material and time consuming. In comparison, the enzymatic methods are scalable and do not result in as much loss of the sample material, but exhibit sensitivity to contamination and bias towards enzyme-specific sequences (leading to a poor representation of the sample). In the end, a properly degraded DNA is a set of fragments distributed along the Gaussian curve with a mean in the desired fragment sizes' range.

These fragments undergo adapter ligation. The particular ligation procedure is determined by a 'library preparation kit' used in the sequencing experiment. Such a kit contains reagents and adapters and optionally targeted panel, ultimately dictating how ligation, amplification and validation steps are performed. In general, in a

process called end repair, the ends of the fragments are tailed with platform-specific 'blunt ends' (synthetic oligonucleotides containing 5′ and 3′ groups and often barcodes for identification). In single-end sequencing, the read is read only once (in 5' to 3' direction). In paired-end sequencing, both ends can be utilized, producing two (forward and reverse, complementary) reads with known distance between them and thus aiding alignment algorithms.

```
 Forward read: 5′ TTTTTTTCTGAGGCAAGTCCCACTCTCTTGCCCAGGCTGGAG 3′
(Reverse read: 3′ AAAAAAAGACTCCGTTCAGGGTGAGAGAACGGGTCCGACCTC 5′)
 Reverse read: 5′ CTCCAGCCTGGGCAAGAGAGTGGGACTTGCCTCAGAAAAAAA 3′
```

In the case that a significant portion of fragments is too long or too short for a particular method of sequencing, a bioinformatic analysis (assembly, mapping, etc.) may suffer from low read coverage of that portion or from unsuccessful addition of the adapter sequence onto the end of the read. To circumvent biased distribution of sizes, the adapter-ligated fragments are size-selected, or separated by the desired size from all others. Such a selection is carried out via gel electrophoresis method (where the naturally negatively charged fragments move towards the positively charged end of the electrified lane with gel, forming a ‘DNA ladder’) or magnetic beads method (where beads bind to the fragments according to the ratio of bead suspension to the solution, and are attracted by a magnetic field).

The optical and semiconductor-based detectors in the MPS instruments are not able to receive sufficient signal from single molecules, and so require an amplification of the DNA material. It is possible to amplify and then sequence only a selection of fragments, apt for a particular research question or a budget. In that case, a set of verified biotinylated oligonucleotides from a kit is utilized to target complementary fragments (binding to, or hybridizing with them), wash away the unbound ones, and amplify the elution (perform target enrichment[24]). The oligonucleotides (oligos, probes) can be either of mRNA or cDNA origin, while a set of them can belong to exonic regions (for whole exome sequencing, WES[25,26]) or a panel of relevant genes.

The amplification is conducted in several rounds and cycles via polymerase chain reaction (PCR)[27,28]. Within one cycle, consecutively, the initial double-stranded DNA is denatured (as hydrogen bonds break), the DNA primer is annealed to the 3' end of each strand, and the complementary DNA strands are synthesized (elongated, extended) by the polymerase enzyme using the deoxynucleoside triphosphates (dNTPs). The primers and the polymerase are selected beforehand, and added to the solution along with dNTPs. The reaction effectively duplicates DNA strands in the solution, thus reaching billions of copies by the end of ~20th cycle, and trillions by the end of ~40th cycle.

According to the reviews of the evolution and modern state of NGS technologies[29-31], he reading of the amplified DNA is achieved either via ‘sequencing by synthesis’

(SBS) or ‘sequencing by ligation’ (SBL). SBS of the discussed ‘second generation’ sequencing methods comprises two technologies:

- Cyclic reversible termination (CRT), where the immobilized DNA strands are DNA-polymerase-supplemented with fluorescent nucleotides of particular colors, imaged with TIRF (total internal reflection fluorescence) detector, and washed with the removal of the fluorescent nucleotides with the strand complements they are attached to; all this cyclically repeated. The method is utilized by Illumina[32].
- Pyrosequencing, where the beads with the DNA strands on them are placed into PTP (PicoTiterPlate) wells along with the beads with catalytic and bioluminescent enzymes on them; after that, the wells are filled with dNTPs of a particular type, which are added to the strands by the DNA polymerase that is also present in the wells; the addition of a dNTP causes a pyrophosphate reaction resulting in the release of light, captured by the CCD (charge-coupled device) detector. After the washing, the cycle repeats. The method was utilized by 454 Life Sciences (on its own and later as part of Roche) until 2016, when the production of instruments was abolished[33,34].

In SBL, the DNA strands hybridize with oligonucleotides with one or two ‘interrogation bases’ and fluorescent dyes, which are ligated to the template with universal primers. After washing of the leftover oligonucleotides, the ligated ones are imaged and identified. The oligonucleotides are then cleaved of the dye, and the cycle repeats. The method is utilized by Thermo Fisher Scientific for their SOLiD sequencing instruments[35,36].

The ‘third generation sequencing’ methods are able to capture whole DNA molecules, hence they are primarily called ‘single molecule sequencing’ (SMS) methods[37]. These still relatively new and unused technologies bring kbp-long reads and increased sequencing speed at the cost of lower accuracy and scalability. For instance, Pacific Biosciences (PacBio) applies a ‘real-time’ variant of SBS[38], while Oxford Nanopore Technologies (ONT) relies on the detection of changes in the ionic current that occur when the DNA strands pass through the nanoscale pores[39]. This data, however, is not used in this work.

Genomic analysis methods and implementations

Overview of workflows

Efficiency and accuracy of the alignment, variant calling and interpretation is sensitive to the systematic errors and biases in the NGS data. The widely recognized issues here are the quantity of the data and the errors induced at various stages of obtaining and processing (sampling, sequencing, mapping) of the data. Because of these issues, the demand for the sophisticated methods of the analysis is on the rise.

In a typical NGS processing workflow, the raw data from the sequencer passes through several pre-processing steps, before a detection algorithm for a particular variation type (dedicated to finding particular types of errors) would start identifying likely variants. Pre-processing includes quality control of the reads (trimming bad quality reads, marking duplicates from library preparation, adjusting quality score observations) and their alignment (mapping the reads to the reference, sorting the alignment files according to the reads' positions) or de novo assembly. During the alignment or de novo assembly, the reads are positioned to overlap each other and assemble contigs (contiguous sequences representing consensus regions). The task of de novo assembly is notably more difficult than that of the alignment: the alignment relies on the index of the reference genome, whereas de novo assembly is used to obtain first versions of reference genomes. Neither can be considered a solved problem, but a number of established methods is used in production environments to-date (discussed in Chapter 5). The choice of the aligner may depend on the source data (DNA or RNA, genome or exome), the types of variants to call (short variants, structural variants), the variant caller(s) used in combination, and any existing best practices. As an example of the latter, the Broad Institute proposes certain workflows for their widely used software suite GATK (Genome Analysis Toolkit)[40,41]. Other developers, too, might suggest datasets, parameters and other tools to use with their software.

The variant detection differs for germline variants and somatic variants from the tumor and hence the genomic pipelines utilize either germline or somatic variant detection methods. Tumors are known to be heterogeneous, consisting of cell subpopulations with independent novel somatic mutations. As a consequence, the frequency of their appearance in the data tends to be significantly lower than the frequency of the germline variation. In addition, in the somatic variant detection a matched normal sample (from the same individual) is utilized to distinguish between somatic and germline variants. This work, however, focuses on germline analysis and features germline variant callers.

Variant calling and variant filtering

Variant calling is, in essence, a calculation of the difference between two or more samples - with one being of interest (e.g. from a patient) and another a reference. Such a calculation can be as simple as counting alternate alleles at the inconsistent positions. However, the biases discussed above make this approach, at best, inefficient. Modern variant calling tools such as GATK, VarScan, SNVer, Strelka2 utilize bayesian, heuristic, frequentist approaches for inferring correct statistics about the variants. Moreover, reads can carry or help to derive complementary information such as base and mapping quality scores, strand orientation, GC content (proved to be limiting sequencing capabilities[42,43]), distances and other metrics that can influence the confidence of the call. If the variant detection method is designed to be permissive, such criteria are helpful in filtering out false positive (FP) variants.

The filter decision boundary has, so far, passed through three evolutionary stages. The hand-crafted decision boundary, or 'hard filtering', is based on the evaluation of the above criteria made by biological experts. Whenever some of their values reaches a specified threshold, the corresponding variant is classified either as true or false positive. This linear approach, however, affects other variants as they may have different suitable values: a portion of the true positives will be either removed from or included in the final callset together with the false positives. Hence, currently this approach is considered crude and is resorted to only when the variant callset is underpowered, containing annotations with values that are not computable by more sophisticated methods.

The more sophisticated methods, in turn, employ probabilistic models that rely on the expert-made criteria utilized as features. The decision boundary is then described by these models, enabling the selection of variants by their fitness to multiple variant profiles. As an example, the two competing variant callers GATK and Strelka2 depend on different machine learning methods (Gaussian mixture model (GMM) and random forest, respectively) to improve the accuracy and consistency of their variant callsets, yet both apply feature sets and trained models to do so. Still, this kind of approaches has limitations, as discussed in Chapters 3 and 6.

The state-of-the-art mode of variant filtering eliminates the limits of human-defined criteria, instead focusing on architectures of artificial neural networks (ANN) that train models on real data. This way, the decision boundary is modelled with features that are learned rather than explicitly set. Several variant detection tools utilizing deep neural networks (DNN) have been published already. The capability of these novel tools to be incorporated into existing variant calling pipelines is yet to be determined.

Interpretation of genetic variation

Variant calling allows researchers to select and identify various types of genetic variations, and eventually perform downstream analysis, from biological annotation to clinical annotation and association with phenotypes. The variants in the callset may have specific functional significance such as encoding a change on a protein or a contribution to a gene; methods of computational prediction of these effects such as SnpEff or Ensembl VEP are often applied to novel variants[44,45]. The variants can also be characterised by passing them through the relevant genomic databases, such as dbSNP and ClinVar[4,46]; annotated, the variants can be arranged according to their clinical significance[43,47]. With known clinical association, the variants can give insight to the genetic architecture, inheritance patterns[48], pathogenesis of complex genetic diseases[49], and, eventually, aid medical evaluation[50,51]. A reliable variant detection is increasingly important for these analyses, as the callsets are expected to be both specific and sensitive[52-54].

Chapter 3: Probability and Machine Learning in Variant Calling

This chapter reviews probabilistic and machine learning methods in application to variant calling. It starts with the application overview and continues to the algorithms and models utilized for the task. The methods are designed to rely on the outcome of another and so described consecutively. Some of the commonly used terms and concepts are straightforward and explained briefly; others require a formal definition to be given separately. In the end, the limitations of these techniques are discussed.

Application overview

Statistical and probabilistic modeling play a key role in modern variant calling. The first problem that requires probabilistic methods is discovering the positions of interest (the ‘active regions’) and candidate haplotypes. For any particular position, the probability of containing a variant is calculated using the data from the alignment. Next, the very problem that genotyping (calculating genotype likelihoods) methods are designed to solve brings the uncertainty: given the position of a base in the reference genome where the base is not completely concordant with the alleles in the aligned reads, and given that the reads may have errors and biases encoded in them, what is the true genotype at this position? To make the decision, the majority of genotyping methods use Bayesian inference, but frequentist, heuristic, and machine learning methods are applied as well or in combination. Finally, variant filtering solutions rely on decision boundaries that are modeled with features that are determined by the human experts.

Accordingly, variant callers of all kinds encompass multiple scores and likelihoods in their execution phases. Here, the emphasis is on the approach conceived by the Broad Institute (GATK HaplotypeCaller + VQSR), due to its consistent performance and a potential neural network based successor (discussed in Chapters 4 and 6), as well as widespread use and an established community.

Methods and implementations

Genotyping with probabilistic models

Genome Analysis Toolkit (GATK), being a toolkit, consists of more than 200 different tools for quality control, read and variant manipulation, variant discovery, filtering, annotation and evaluation (considering major version 4, and including tools that are in beta or experimental state and tools that belong to Picard, a quality control toolset that is built in GATK). Germline short variant discovery starts with one tool: HaplotypeCaller[55]. HaplotypeCaller, as a collection of tasks, accepts alignment and reference sequence data, finds 'active regions' in the alignment, finds haplotypes and their likelihoods therein, finds genotypes with Bayes' rule, and outputs the most likely ones into a variant callset (Chapter 5). The program progresses step-wise and locus-by-locus.

Step 1. To determine 'active regions', at first, it sifts through pileups of each position and, based on any alternative allele count in a given variant context (SNVs or indels or portions of unsuccessful alignments) as evidence and heterozygosity rates of SNVs and indels as priors, assigns a 'profile state' probability value. Resulting 'raw' probability profile of every one-residue loci ranges from 0.0 to 1.0, with the default threshold of 0.002 for being 'active'. This threshold, along with the default standard deviation (17 bp), is then used to create a Gaussian kernel based band-pass filter in order to smoothen (convolve) the probabilities over the regional intervals. The 'active region' is always extended by 100 bp on each side.

Step 2. Each 'active region' contains a number of candidate haplotypes, the best (most supported) of which are selected via assembling a De Bruijn-like graph of the reference sequence in region's coordinates and matching each of the reads with each of the graph's paths. In such a graph, the more consecutive read segments (k-mers) correspond to the adjacent vertices, the more weight the edge will have. The resulting graph is adjusted, or pruned (Figure 1): the paths with edges of weight lower than threshold (2 by default) are removed, the paths without a terminal k-mer (a 'dangling tail') are either removed or merged to the reference path via the Smith-Waterman alignment (SWA) algorithm[56]. After pruning, the remaining paths define the candidate haplotypes. Each of these haplotypes is then realigned to the reference by the SWA, producing CIGAR strings (Concise Idiosyncratic Gapped Alignment Report, or an indication of the amount and order of matches, mismatches, insertions and deletions of the sequence in relation to reference) and pinpointing the location of the candidate variants.

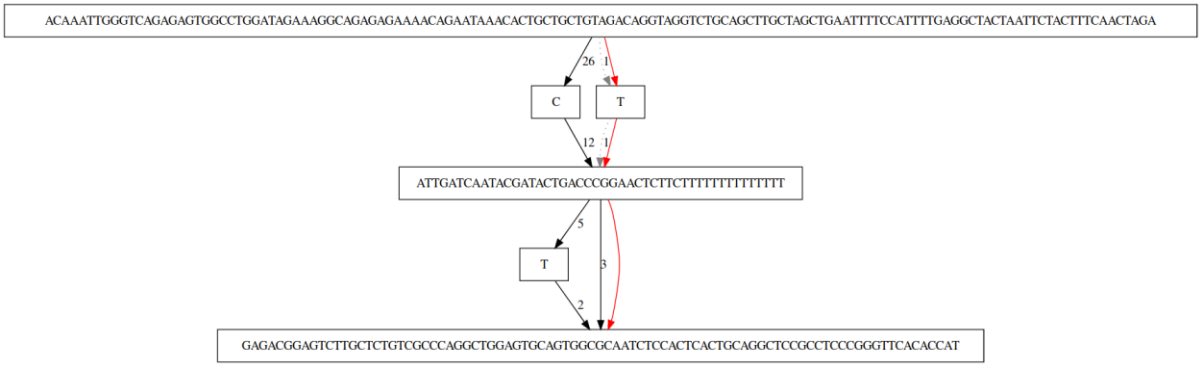


Figure 1. An example of a local haplotype reassembly. Here, the red edges represent the reference path, the grey and dotted represents the pruned one, and black represent the reads' contribution. This graph was collected using a "--graph-output" argument intended for debugging.

Step 3. The support for these haplotypes is determined by the pairwise alignment of each read against each candidate haplotype (Figure 2). The pair-HMM (pair Hidden Markov Models)[57,58] algorithm calculates a matrix of *transition probabilities* (TP) of the alignment states (*transiting* to a Match, an Insertion, or a Deletion), individually characterized as

$TP = GP$ or $TP = 1 - x * GP$, with $GP = 10^{\frac{-q}{10}}$,

where GP denotes a gap penalty that is based on q, which is either a gap extension penalty (a constant of 10) or an insertion or deletion gap open penalty (a constant of 45 or a base quality present in the recalibrated alignment), and the subtraction and x depend on the alignment state of a particular position. The exponential function is antipodal to the Phred Quality Score's logarithmic function.

For any particular read and haplotype combination, the alignment states are computed recursively, producing also a matrix of *emission probabilities* (EP) of all of the positions in a read against all of the positions in a haplotype, individually characterised as

$EP_{non-matched} = GP/3$ or $EP_{matched} = 1 - GP$,

depending, accordingly, on whether a corresponding TP showed a match. EP describe the observed probabilities of the read-haplotype alignments. Once aligned, the matrix of likelihoods is summarised to a total likelihood score of a read-haplotype pair. These scores, in turn, form a matrix of likelihoods of all of the reads against all of the haplotypes.

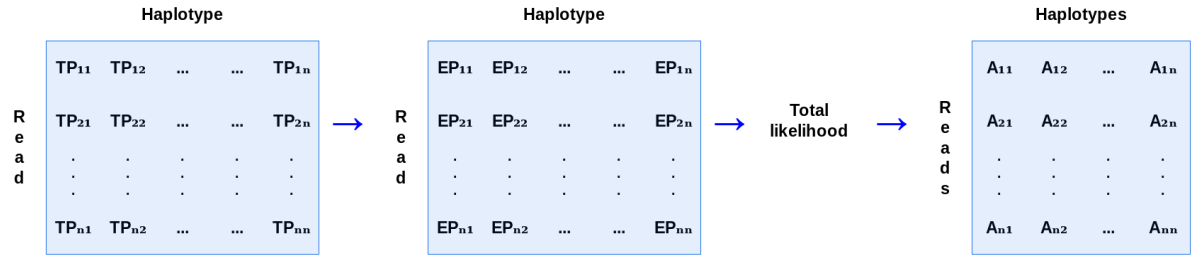


Figure 2. The pair-HMM workflow with matrices of probabilities. The matrices are loosely based on GATK workshop materials.

Step 4. The acquired haplotype likelihoods are then derived to allele likelihoods. For every potential variant (extracted from CIGAR), there is a number of haplotypes supported by any particular read, of which the highest-scoring haplotype gets to represent the allele of the chosen read (Figure 3).

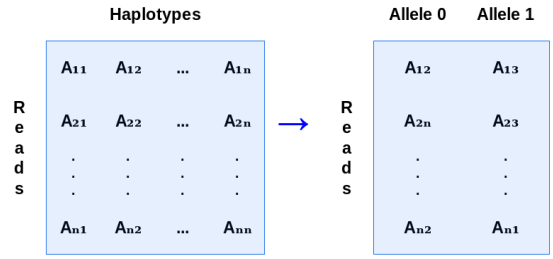


Figure 3. Transforming haplotype likelihoods to allele likelihoods. The elements chosen for the example are loosely based on GATK workshop materials.

The knowledge of haplotype likelihoods associated with chosen alleles is crucial for determining the 'raw genotype probability' (called so by Poplin et al.[55], in reality describing the conditional probability of the read given the genotype) and, consequently, the posterior genotype probability of every variant (the goal of the HaplotypeCaller). The posterior genotype probability is calculated with the Bayes' rule:

$P(G_i|\, R_j) = \frac{P(G) * P(R_j|\, G_i)}{\Sigma_i\, P(R_j|\, G_i) * P(G_i)}$, where

$P(G)$ is the prior probability of a genotype that is either default ($\frac{1}{2 * num\_samples + 1}$) or user-defined, and therefore made flat (identical) for all possible genotypes,

$P(R_j|\, G_i) = \Pi_j(\frac{P(R_j|\, A_1)}{2} + \frac{P(R_j|\, A_2)}{2})$ is the mentioned probability of the read with that genotype, that relies on the allele likelihoods,

and the denominator sums all possible genotypes of these alleles.

By design, the HaplotypeCaller's approach to draw genotype likelihoods is sensitive rather than specific: it looks for every candidate region, haplotype or allele, but it does not account for potential sequencing and initial alignment artifacts and errors, apart from the option to undervalue the repetitive sequences' indel base qualities to make the pair-HMM more stringent against such regions. Both repetitive and non-repetitive indel, as well as SNV calls may or may not represent actual variants, and thus are officially not suitable for a comprehensive downstream analysis (the validity of that statement is evaluated in Chapter 6). Fortunately, HaplotypeCaller leaves multiple 'annotations' as auxiliary information (reporting of number of chromosomes, depth of coverage, mapping quality, allele frequency, strand bias, as well as just computed genotype likelihoods) for every discovered variant, making the filtering of false positive variants possible.

Variant filtering with machine learning methods

Variant filtering, as mentioned in Chapter 2, can be performed by cutting off variants which annotations did not pass the empirical threshold. For example, any variant with 'MQ' (mapping quality) annotation lower than 40 or 'FS' (Fisher's Exact Test for alleles' support on both strands) annotation greater than 60 could be discarded. It stands to reason that different variants in the callset exhibit different values in their annotations, albeit without an extensive deviation.

Therefore, even if the threshold is optimal, the variants with annotation values close to the threshold could be classified unjustly: both true and false positive variants would reside on both sides of the threshold. If the threshold favors keeping more true positives, the filter keeps more false positives along; if the threshold favors the removal of false positives, the filter clips true positives as well. Moreover, if two or

more filters are applied to the callset, they can be contradictory and, as a result, too loose or too stringent. For this reason, the 'hard filtering' is considered inefficient and expensive in terms of manual selection of appropriate annotations and thresholds.

To remove unwanted variants from the callset more efficiently, their selection should be non-linear, with effect from multiple, if not all annotations taken into account. In GATK, a proposed solution is to consider a combined effect of these annotations, modeling 'annotation profiles' of true positive variants. The *VQSR* method, part of the GATK package, implements it as a machine learning estimator, 'learning' these profiles from the datasets of known variants and then using them to label new variants[41,59-61]. Because its description requires an understanding of fundamental machine learning concepts, a primer on machine learning is supplied.

Machine learning

Machine learning is an algorithmic approach for finding robust solutions to problems, in which the model is trained on the existing data, instead of being defined as an exact algorithm. Such models are able to extract patterns from heterogeneous, multi-dimensional datasets, and use these patterns to characterize the new data. Machine learning algorithms comprise several branches, each of which is applied to the most suitable tasks: *supervised learning* includes classification and regression algorithms; *unsupervised learning* includes dimensionality reduction and clustering algorithms; *semi-supervised learning* algorithms may consist of a combination of the previous two; and *reinforcement learning* algorithms tackle tasks that do not have a clear definition but do provide a feedback, such as playing a game. In variant calling, as of today, only supervised and unsupervised methods are known.

Learning from data means *training* on the incoming data until the model is able to generalize the predicting output to the new, unseen data. In supervised learning, the training data is labeled: its *instances* (observations, examples) are associated with a description value, the likes of which the model is supposed to predict. In unsupervised learning, there are no labels: the algorithm is set to find the similarities or dissimilarities between the instances of data that it is fed. The instances consist of the *features*, which can be categorical, continuous or mixed values; the type-specific data is analyzed by type-specific algorithms and techniques.

The central idea of training is based on minimizing an error measure, thus maximizing the correctness of the prediction. The way it is obtained is fundamentally different between the branches: with labels, it is the *training error* (loss function, defined as a measurement of loss from a prediction being wrong, e.g. as a mean squared error (MSE)) and the *testing error* (generalization error); without labels, the only alternative is the *general criterion*, such as a distance measure.

For supervised learning, its labeled dataset is divided into the two or three separate sets: the training set, the validation set and, possibly, the testing set. *The training set* is used to fit (update) the parameters (e.g. coefficients, weights, biases) of an

algorithm, by predicting the label for every available instance and computing the loss function. The error is minimized, or optimized, via the *gradient descent* algorithm: knowing the current value of the loss function $J(\theta)$ and the parameter θ, a partial derivative is able to find the gradient (the extent) to which the parameter value should be changed:

$$\theta = \theta \ - \ \alpha * \frac{\partial J(\theta)}{\partial \theta},$$

where α represents an arbitrary learning rate that defines the speed of the gradient, and that depends on the particular implementation.

The validation set serves as an indicator of the trained model's ability to generalize, i.e. predict as accurately on the unseen data; at this point, a researcher may want to optimize the hyperparameters (tunable aspects such as the learning rate for minimization algorithm or the amount of trees in a random forest) of the model to improve the fitness. Here, *cross-validation* techniques may be utilized to iteratively partition the initial dataset and obtain an averaged estimation of fitness; a *nested cross-validation* may be utilized to select the best-performing hyperparameter set. Finally, *the testing set*, that is in good practice completely independent from the training, evaluates the final model (or models) performance.

With unsupervised learning, the structure of the data or the relationships between the instances are decided without any external criterion. The clustering algorithms calculate the metrics (distances, variances) for instances, assign the clusters upon them, and adjust with each new iteration, until there is nothing left to adjust (until *the convergence* is reached). As a result, the distributions, densities, similarities are learned from the given features, and these properties correspond to the user-defined hyperparameters (like the amount of clusters or instances in a cluster).

Training on the noisy or biased data or with wrongly chosen hyperparameters may lead to unwanted results, such as *overfitting* ('memorizing' the prediction instead of generalizing) or *underfitting* (ignoring the actual structure of the data by rendering simplistic prediction). Accordingly, the methods of dealing with these issues involve manipulation of the data or the model. To avoid overfitting, the model is *regularized*, by applying a penalizing technique (such as L1 Lasso or L2 Ridge) to the parameters or reducing the amount of the selected features. To avoid underfitting, the existing regularization could be reduced, or more features could be selected or engineered.

VQSR

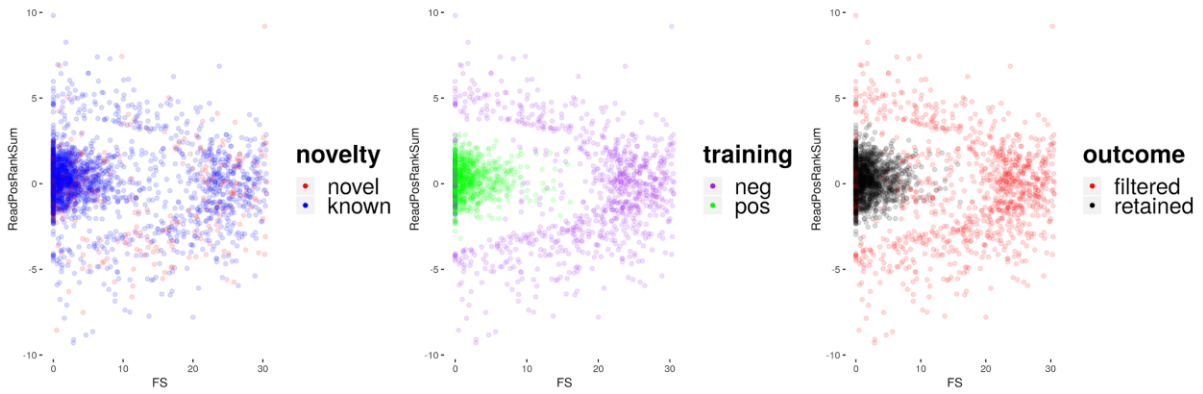
The ‘raw’ callset, obtained with HaplotypeCaller, contains both unseen variants that are supposed to pertain to the analyzed sample or individual, and variants that can be found in databases of validated and recognised variants (more on databases in Chapter 5). The latter variant sites can be categorized as

- true sites, validated high-confidence variants that will be used in training;
- non-true sites, partially true and partially false variants that will be used for training, too;
- non-true sites, ‘known’ variants that will be used for additional metric stratification rather than training.

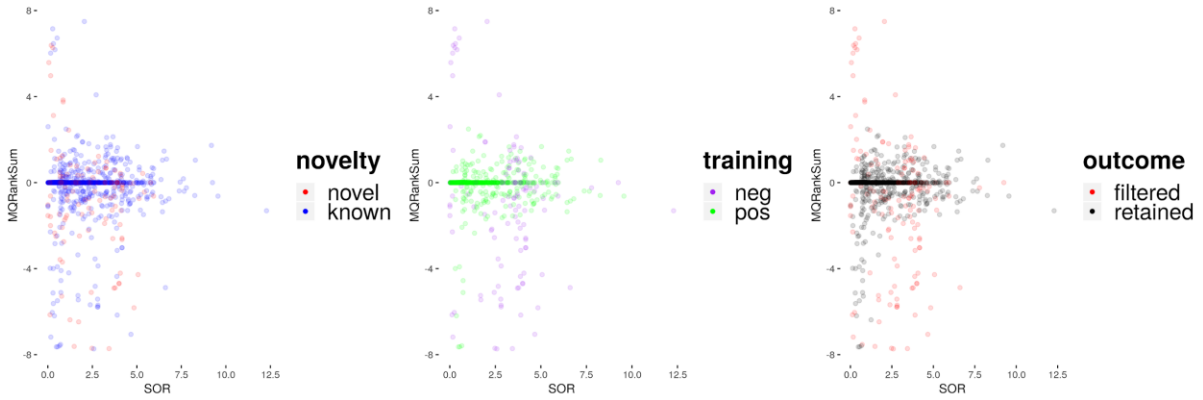
The datasets of human variants, adjusted to several widely-used human genomes, are supplied by the Broad Institute. The method utilizes the datasets of these variants (‘resources’), separately for SNVs and indels. Accordingly, the annotations of the called variants with such associations are used to train positive and negative Gaussian Mixture Models (GMM) for SNVs and for indels. The Gaussian mixture distribution here is chosen due to its ability to represent a realistic dataset, that may consist of more than one Gaussian distribution, as in this case (Figures 4a, 4b).

The variational Bayesian algorithm learns the GMM cluster parameters and fits the other variants in that callset to these GMM clusters, to estimate them as close to being true or false with a VQSLOD (Variant Quality Score Log-ODds) score and a corresponding tranche (designating variant’s sensitivity to the true sites). The hyperparameters for the variational Bayesian algorithm such as the number of Gaussian distributions per model, and shrinkage and dirichlet parameters (used for computation of multivariate Gaussian distribution) are made default, but the user can change them with the tool’s arguments. In the end, the VQSLOD-tranche threshold is applied, and the false positive variants filtered according to it.

It may happen that annotations of a variant are too divergent, in which case they will not be used in the GMM. VQSR accepts additional arguments for tweaking, such as the standard deviation threshold for these annotations. In a general case, VQSR tools accept input callsets, resources, resources’ prior probability notations, and annotation list for modeling as arguments. The method often relies on tailoring settings according to the needs of a particular project; it has been noticed that it may be prone to underfitting or overfitting the data[62].



(a)



(b)

Figure 4. An example of the recalibration plot from the VQSR output, showing the distribution of **(a)** SNV variants along FS/ReadPosRankSum annotation scales and **(b)** indel variants along SOR/MQRankSum annotation scales; other annotation dimensions are available. The 'novelty' plot describes the distribution of 'novel' and 'known' variants. The 'training' plot describes the variants in the positive and negative GMMs. The 'outcome' plot describes how the variants were filtered.

Chapter 4: Deep Learning in Variant Calling

This chapter extends the previous chapter on machine learning with the concept of deep learning. It starts with the origins of neural networks and continues to the range of architectures and methods used today. Then, several neural network based variant detection tools are discussed.

Application overview

At least five variant calling and filtering tools that utilize one or another kind of neural network architecture are already available. The claims about their performance compared with conventional machine learning models are tested in Chapter 6; here, the deep learning techniques that these tools rely on are examined.

Deep learning is a division of machine learning focused solely on multilayered neural networks. A neural network is composed of processing nodes (‘neurons’) organized in layers, receiving the input signals from neighboring nodes, computing the weights and passing the result (net input) to the next layer, making the network learn or process the features. A particular ordered combination of layers - a model - defines how training or inference can proceed. The models may differ in the amount of layers ('depth' of the network), layers’ dimensionality ('width' of the network), activation function, and optional regularization and constraint functions; each model employs an appropriate loss function that is optimized using an appropriate minimization algorithm.

During the training, the parameters (weights, thresholds) of the connections between the nodes are assigned as zeroes or randomly and then adjusted until the model is trained. As with machine learning approaches discussed in Chapter 3, the input (feature) signals can be used as a learning material to predict the output (label) signal. The features, however, are not (necessarily) engineered by a human expert; they are extracted from the dataset. With every iteration, as the cost function estimates how far the prediction is from the true values, its value is supposed to decrease, nearing zero. This optimization is performed by a training algorithm, usually stochastic gradient descent, where the gradient itself is procured by the backpropagation algorithm.

During the inference, the nodes in layers, after receiving their part of the data, multiply it with the assigned weight and sums with the analogous result from another connections, until certain threshold is reached, upon which the data is passed on again. The weights and the thresholds are only changed (‘trained’) during the training process, and are fixed for the inference.

Feedforward neural networks (FFNN), recurrent neural networks (RNN), autoencoders (AE) and deep belief networks (DBN) constitute major types of

networks developed and applied today, with many more subtypes designed for more specific tasks.

In the next section, the general methods behind neural networks are discussed.

Methods

Architectures

As the name implies, a feedforward neural network is unidirectional, passing the data only forward - to the layers above the observed one. Therefore the nodes in such networks receive the data from the nodes beneath. Feedforward neural networks, originating from the simplest of neural network architectures, multilayer perceptron (MLP), are often initially applied to various kinds of tasks requiring analysis of high-dimensional data.

An MLP only consists of fully connected (FC) layers, where every node has a connection with every node of the layer above, apart from the output layer (Figure 5a,b). The nodes of the input layer feed the data they receive to the nodes of the following layer, where the data from every input node is multiplied by the current weight, and summed with such multiplications from every other input node. The result is passed through the activation function ('activating' the node and passing the value further upon a condition, such as removing negative values in ReLU). Between the input and output layers the hidden layers are hidden. An MLP may have an arbitrary number of hidden layers, modeling complex, nonlinear functions; however, too large number of connections could set limits to training[63,64].

Another subtype of a feedforward network, a convolutional neural network (CNN), has more complicated structure: *convolutional* layers pass their computed weights to pooling layers, to new convolutional layers, and in the end, to FC layers again (Figure 5c). As CNNs are often used to process multidimensional, large-scale data such as images, the layers of the network, too, must become multidimensional (usually 3-dimensional), by creating multiple filters (kernels) of the fitting depth and particular width and height dimensions. The nodes in such layers only connect to a subset (a neighboring region) of the layer above, due to the exponentially larger amount of weights and inability to scale and generalize well otherwise.

The input convolutional layer slides over the input tensor (a data structure that holds multiple dimensions), convolving (merging by multiplication of values in a given filter and the input, and summing the results) the features into a feature map. The maps stack onto each other, forming the input for the next layer from the higher to the lower level features. The convolutional layer is followed by the activation layer, which is followed by the pooling layer (that reduces the size of the input by picking an average or maximum value in another sliding window). The output of the network is made by the FC layer or layers, producing vector or vectors with label values corresponding to the ones the network was trained with[65,66].

Recurrent neural networks, now divergent from feedforward neural networks, allow *feedback* connections from the output to be added to the input, and therefore process the inputs in a looped, *sequential* manner. The number and the location of the feedback loops can be arbitrary. Inside the loop, RNN updates information, but not the parameters, creating a series of hidden states with memory of the previous timesteps (loop operations are considered individual passes). This memory, however, has a known property of *vanishing* over time (vanishing gradient problem), with rate depending on the activation and optimization functions used[66,67]. A subtype of RNN, a Long short-term memory (LSTM) network was designed to control the gradient growing or shrinking via the 'gate' units associates with nodes, that represent the possible 'cell states' (Figure 5d). The gates determine, based on weights and the hidden state, the portion of memory of the previous state that gets to 'input' the node, and 'forget' or 'output' the current state to the next node. Thus, the network remembers relatively long sequences, and can learn and predict relevant sequence relationships[68,69].

Autoencoders are used to reconstruct the input, first encoding them with the compression, and then decoding the compressed representation. This is done without any labeled data, by calculating the reconstruction error. With one or more hidden fully connected or convolutional layers, autoencoders reconstruct the data, leaving only the most important, meaningful features (Figure 5e).

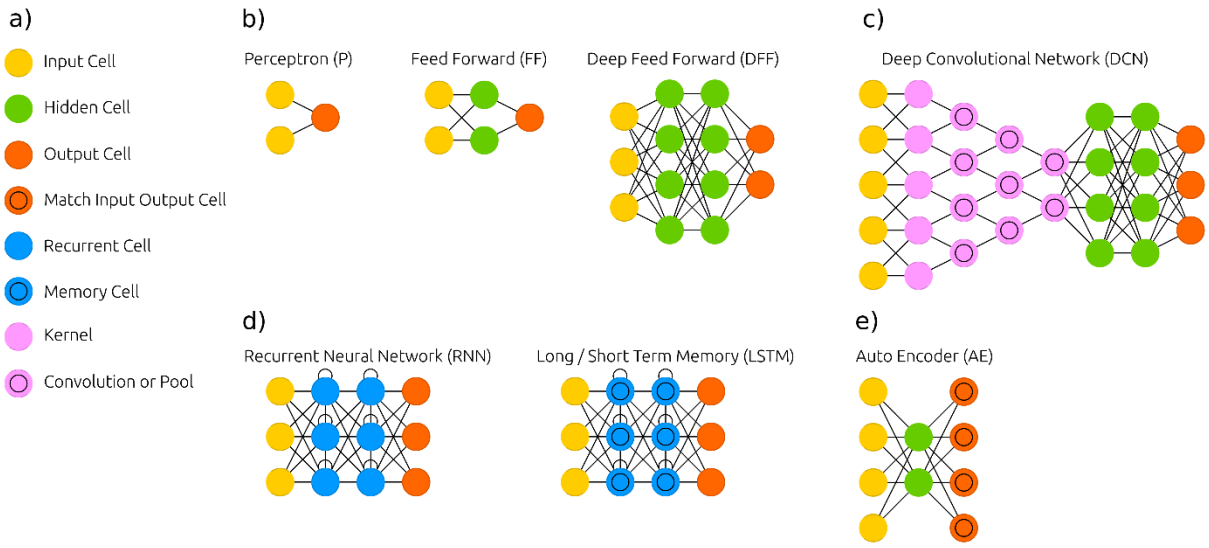


Figure 5. A selection of neural network architectures from Neural Network Zoo[65]. Here, a) shows possible node (here, 'cell') types of the included architectures; b) shows basic feedforward architectures, with MLP being the second and the third ones; c) shows CNN architecture, dubbed as DCN; d) shows RNN and subtype LSTM architectures; and e) shows the architecture of an autoenconder.

Neural networks are flexible in that they can take various forms and combinations of different layers. This property has been utilized by many authors of methods, including the authors of the observed variant calling methods.

Relevant concepts

In the previous section, several concepts have been introduced, that have not been explained in Chapter 3, or need further elaboration.

The training of a neural network is an iterative process: the algorithms pass through a multitude of epochs that are processed in a number of iterations. In one iteration, exactly one batch from a dataset is processed; the size of the batch indicates the amount of instances contained within it. Often, the batch input is *normalized* by its mean and standard deviation (standard score) for the hidden layers, improving the training speed, initialization and regularization[70].

The process starts with the initialization of weights, which will be used from the very beginning of the training. The weights can be set randomly, or using a distribution-generating *initialization algorithm* that complements the activation function, so that the rate of convergence of the network is high. Then, for feedforward networks, three iterative steps are performed:

1. The layers are activated with their respective *activation functions*, i.e. the weights are recalculated and 'forward propagated' up to the output layer that makes the prediction. Accordingly, an activation function determines, by computing the parameter with a condition, whether its node should pass the parameter to the layer above. The function is nonlinear, so that it would not result in only a linear model. Some of the activation functions are used in the output layer, i.e. before computing the loss. The particular functions are selected as corresponding to the task, such as classification or regression, and with regard to the position of the layer and to the potential vanishing gradient[66]:
    - The Sigmoid function, defined as $f(x) = \frac{1}{1+e^{-x}}$, produces values in the range of [0:1] that form a steep curve with most of the changes in the input range of [-2:2]; it could be a choice for classification, but also for the output function in binary classification.
    - The Tanh (hyperbolic tangent) function, defined as $f(x) = \frac{2}{1+e^{-2x}} - 1$, produces values in the range of [-1:1] with even more steep curve than that of Sigmoid. It is more commonly used in RNN than other architectures, especially in LSTM gates due to their efficient convergence.
    - The ReLU (rectified linear unit) function, defined as $f(x) = max(0, x)$, produces values in the range of [0:∞] and is, in fact, nonlinear; it is able of zeroing the negative values and excluding the node from the computation chain, making it more efficient. ReLU may be seen as a more universal approximator, therefore many deep learning models rely on it.

- The Softmax function, defined as $f(x_i) = \frac{e^{x_i}}{\Sigma_j e^{x_j}}$, produces a vector of values in the range of [0:1] totaling to 1. It serves as the output function in multi-class classification.

2. The appropriate *loss function* is calculated, registering the error of the prediction (Chapter 3). Two types of loss functions may apply:
   - For the general case and especially for classification problems, the cross-entropy (negative log likelihood) loss function is suitable. The function tends to stay convex and is thus often preferable for gradient descent optimization. The cross-entropy measures entropy, or distance between two probability distributions, the true distribution and the predicted distribution:
     $Cross\_entropy(y, \hat{y}) = \frac{1}{m}\sum_{i=1}^{m} L(x^{(i)}, y^{(i)}, \theta)$,
     where $L(x, y, \theta) = -\log p(y|\mathrm{x}; \theta)$
   - For regression problems, where the prediction values may or may not stay within the range of [0:1], a sum of differences error such as Mean Squared Error (MSE) is more applicable:
     $$MSE(y, \hat{y}) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y_i})^2$$
3. The gradient descent algorithm aims to reach the minimum of the loss function in a steep manner, causing the network to converge rather than diverge (Chapter 3). Most modern neural networks utilize a variation of the stochastic gradient descent (SGD) algorithm, which picks random (stochastic) examples (in minibatches) from the training set rather than processing the whole training set in each iteration. The loss function is optimized by calculating the adjustment of the weights needed for a current gradient descent step for every layer backwards, i.e. *backpropagating* the error down the network.
   - The backpropagation is done through a chain of derivatives of the loss function with respect to the weights between the output nodes and their parent nodes down to the first input nodes. Accordingly, the derivative of an output is a composite function of the derivatives of the weights that led to the output:
     $$\frac{\partial z}{\partial w} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w} = f'(y)f'(x)f'(w) = f'(f\big(f(w)\big))f'(f(w))f'(w)$$
   - The weights of the network are updated by a gradient, which is an average over each individual derivative. The gradient itself is adjusted by a learning rate hyperparameter, which defines the size and therefore the effect of the step. Its choice may lead to the unwanted consequences, such as reaching a suboptimal global minimum or a high loss function value. The following methods may be used to set or leverage this hyperparameter:

i. Using a default or otherwise fixed learning rate, e.g. with a typical value. Today this is not considered an appropriate method, because it may not correspond to the updates.
ii. The learning rate may be *decayed* over time, which is referred to as the *learning rate schedule*. This involves setting an initial learning rate and a linear decrease over a number of iterations.
iii. Using another hyperparameter to aid calculating the moving average over the gradient history. The moving average also *decays* over time, and the speed of the decay is determined by this hyperparameter. The *momentum* algorithm leaves the learning rate fixed and multiplies the step size following the latest succession of gradients.
iv. Using adaptive optimization algorithms that adjust the learning rates for individual weights according to the condition, such as the scale of the derivative (and weight, by extension), implemented in the AdaGrad algorithm. AdaGrad and its variations with moving average or momentum, such as RMSProp, AdaDelta, Adam are commonly used as optimizers in neural networks.

- The gradient descent may lead the loss function to a position of *global minimum* (globally lowest value) *or local minimum* (locally lowest value), or other *critical point* such as a *saddle point* (neither higher nor lower than a local region). In deep learning, the optimization is considered to be difficult due to non-convex loss function, that may have many global minima and infinite local minima.

For recurrent networks, the backpropagation is turned into the backpropagation through time (BPTT): after the network made the prediction, the loss functions of the repeated parameters of the timesteps are summed, and the adjustment is backpropagated, again, through the timesteps and down the network.

The regularization in deep learning is supplemented by the dropout, where an arbitrary number of nodes of the network are randomly ‘dropped out’ (excluded) in order to hinder overfitting[71]. Currently, it is not recommended to apply the dropout technique to non-fully connected layers[72].

Implementations

The five variant callers utilizing deep learning models are: Clairvoyante from R. Luo et al.[73]; GATK CNNScoreVariants from the Broad Institute[74]; GARFIELD-NGS from E. Giacopuzzi et al.[75]; DeepVariant from the Google Brain team[76]; and VariationAnalysis from Campagne Laboratory[77]. Three of them (DeepVariant, CNNScoreVariants and Clairvoyante) are based on the convolutional neural network (CNN) architecture, a variation of FFNN; the rest rely on a custom kind of FFNN. The CNN callers convert either the aligned reads (DeepVariant, Clairvoyante) or the raw variant calls (CNNScoreVariants) into multichannel tensors, encoding alleles, base and mapping quality scores and read/base properties for the CNN-based classifier. After that, the classifier detects various motifs among these properties and selects the most likely variant positions from the candidate variants. The raw variant calls are also feeding GARFIELD-NGS' neural network, enabling the prediction of a new variant property based on the other properties in a variant call record. The VariationAnalysis framework allows the user to train a combined (with FC and LSTM layers) model or apply a pre-trained model to the pre-converted alignment file, thus predicting genotypes. The models for the prediction have been designed and trained by the developers of each tool, but limited options to retrain the models are offered, too.

Clairvoyante

Architecture. The CNN of the tool is a composed of five layers, beside the input and output: three convolutional layers of different dimensions for feature learning, and two fully connected feedforward layers for making predictions of different types. The activation function of all hidden layers is SELU (Scaled Exponential Linear Units that bring 'self-normalization', derived from ReLU, defined as $f(x) = \lambda x \; if \; x > 0$ OR $f(x) = \lambda \alpha e^x - \alpha \; if \; x \leq 0$, where the parameters λ and α are fixed for known mean and variance)[78]. Aptly, the He algorithm initializes the model. For regularization, the L2 and dropout methods are used. For optimization, the Adam algorithm (variant of SGD) with two distinct learning rates is used.

Training. Clairvoyante was trained on two whole-genome human samples (HG001 and HG002, see Chapter 5 for details[79]), sequenced with Oxford Nanopore, PacBio, and Illumina instruments. Clairvoyante employs a supervised approach and so learns from the labeled data; the Genome In A Bottle (GIAB)[80] truth variant dataset served for this purpose. The samples were randomly partitioned into a training set (taking 90% of the data) and a validation set (10% of the data). No cross-validation is mentioned in the documentation.

I/O. The input layer takes in 'multi-dimensional' (33x4x4) tensors that encode sequence positions surrounding the candidate variants (16+1+16), counts of different alleles on the reads (A, C, G, T), and notations of the supporting source of these alleles (reference, insertions, deletions and mismatches). The output consists of the four vectors, encoding the called variant (either or a combination of A, C, G, T), the

zygosity (homozygous or heterozygous variant), the variant type (either of reference, insertion, deletion, or mismatch), the length of a variant (from 0 to more than 4).

Rationale. Clairvoyante was developed to tackle primarily single molecule sequencing data, that is known to have rather large amount of errors but significantly longer reads (Chapter 2). Additionally, the reads from PacBio are not accompanied by the base qualities. This kind of data sets further limitations to variant calling by the conventional methods. With the advancements of neural networks in analyzing complex data, the authors saw the opportunity to develop a deep learning approach. The network's design may have been influenced by the DeepVariant variant caller that was introduced earlier; nonetheless, the specific architecture was devised from scratch. Clairvoyante's network targets a variant calling task, as opposed to utilizing an image analysis network by DeepVariant.

CNNScoreVariants (GATK4)

Disclaimer. As of the time of writing, CNNScoreVariants is out of beta[81]. However, the technical information remains sparse, consisting only of the developer blog post[74], four similar conference talks[82-85], and the tool's documentation[86]. No article has been released. Some intuition was drawn from the code in the GitHub repository[87].

Architecture. The tool utilizes two different models in its '1D' and '2D' modes, respectively. Other models can be trained and applied using the related tools from GATK, the architectural files and the CNNScoreVariants' arguments. The default '1D' model is composed of seven hidden layers: five convolutional and two fully connected layers. The '2D' model comprises five hidden layers: four convolutional, all considerably wider than that of '1D' model, and one fully connected layer. Before passing to the fully connected layer(s), the convolutional layers are concatenated with a multilayer perceptron (MLP) that is used to compute variant annotation features. The activation functions of both models depend on the usage of batch normalization, which is by default applied to convolutional layers in '1D' model and not in '2D'; the fully connected layers are not normalized. With batch normalization, the activation is linear; without it the ReLU activation is used. All layers are initialized by the Glorot normal initializer. For regularization, the spatial and convolutional dropout is used. For optimization, the Adam algorithm is used.

Training. The models were trained on the data associated with both validated and non-validated callsets. The HG001, HG002 (Chapter 5)[79] and SynDip (from the 'synthetic-diploid' dataset derived from the PacBio sequence data[88,89]) samples were partitioned into the training, validation and test sets (with unknown shares). The truth sets used are from Platinum Genomes (likely only for HG001)[90], GIAB (either or both HG001, HG002)[80], and SynDip (CHM WGS1).

I/O. CNNScoreVariants relies on HaplotypeCaller to create a 'raw' callset and takes it for the input. The variants in the callset are converted into the 'multichannel'

tensors. With the '1D' model, the tensors consist of the alternative allele options, flags for the variant types, input annotations, and reference sequence data. The '2D' tensors add the read data (aligned reads showing the position, strands, mapping quality) to that list. The output labels of the network are formed into a score for each variant, which is a natural logarithm of the true over false prediction of the variant state (SNP / $\underline{SNP}$, $INDEL$ / $\underline{INDEL}$). The score is written in the form of a 'CNN_1D' or 'CNN_2D' annotation in the updated callset (Chapter 6).

Rationale. The CNNScoreVariants' network was designed with the existing GATK workflow in mind. That is, the goal was to complement and potentially replace variant filtering step, currently conducted with VQSR method. Unlike VQSR models, however, CNNScoreVariants models were pre-trained on the labeled datasets, allowing for classification of true variants against sequencing errors. With the architecture able to process comprehensive information about a particular variant in a tensor, and multiple sources of training data, the CNN is supposed to be able to generalize well.

DeepVariant

Architecture. DeepVariant is based on the Inception v2 and later v3 CNN architectures[91], originally developed to solve the image classification problem. In 2014, this kind of CNN marked an algorithmic milestone by achieving accurate prediction on both large and small scales of the image by stacking together 'Inception modules' consisting of multiple filters of different sizes. These modules were acting in place of the commonly used convolutional or fully connected layers. In essence, the filters in one 'Inception module' are convolved and then concatenated together, creating a network inside a bigger network. The original Inception (GoogLeNet) network contained 22 layers overall with 9 double-layered 'Inception modules'[92], while Inception v2 contained 42 layers with 5 triple-layered and 5 quintuple-layered 'Inception modules'. The adapted network incorporates nine modules, called 'partitions'. The exact enclosed layers and filters are not clarified.

The authors of the Inception network series introduced branch layers of 'auxiliary classifiers' with softmax loss, encouraging the convergence during training and regularization (with branch batch normalization or dropout). They have also estimated whether linear or ReLU activation function yields more accurate results, with ReLU performing noticeably better. The authors of DeepVariant do not disclose such details.

The DeepVariant network was initialized with weights from the Inception v2 presented for the ImageNet challenge in 2015. For optimization, it relies on the SGD algorithm.

Training. DeepVariant includes two models, for WGS and for WES data. While not architecturally different, the models have been trained on WGS and WES datasets, respectively. The WGS dataset used to train the model v0.4 (from the publication)

includes 9 samples of the HG001 genome, both public and independent; the current version (v0.7) includes 14 HG001 samples and 2 samples of the HG005 genome (Chapter 5)[79]. The WES dataset includes 78 HG001 samples and 1 HG005 sample. All of the samples have an associated variant truth set from GIAB (Chapter 5)[80]. The datasets had been split by chromosomes: 1-19 chromosomes had been used for the training, and 20-22 chromosomes had been held out for the testing.

I/O. The adapted architecture processes candidate variant's data represented as tensor. The first publicly demonstrated (in the precisionFDA challenge[93]) DeepVariant version encoded the data into three channels: for the read base (A, C, G, T), base quality scores, and mapping quality score, allowing to contain it in an RGB 'pileup image' (Figure 6). In the released versions more channels were added; in v0.7, they encode the strand of the read, and the allele support in the read and in the reference, totaling six channels.

The input layer of the CNN further transforms the tensors, raising their dimensions to the dimensions of the 'receptive field' (221x100→299x299), apparently increasing the accuracy of the model. The output layer assigns one of the three genotypes (homozygous reference, homozygous alternative, heterozygous) to a candidate variant.

Rationale. DeepVariant utilizes a complex, deep and wide neural network with multiple training sources in an attempt to overcome the shortcomings of conventional modeling methods, such as a lack of generalization. It is claimed to not apply thematic knowledge, instead relying on the sophisticated design that is exercised in a spectrum of computer vision tasks. The network does take advantage of a HaplotypeCaller-like algorithm that looks for candidate variants.

The authors have shown that the method is able to generalize to different sequencing instruments (Illumina, Ion Torrent, 10X, SOLiD, PacBio), reference genomes (b37 and b38, see Chapter 5) and even species (human, mouse and rice[94]).

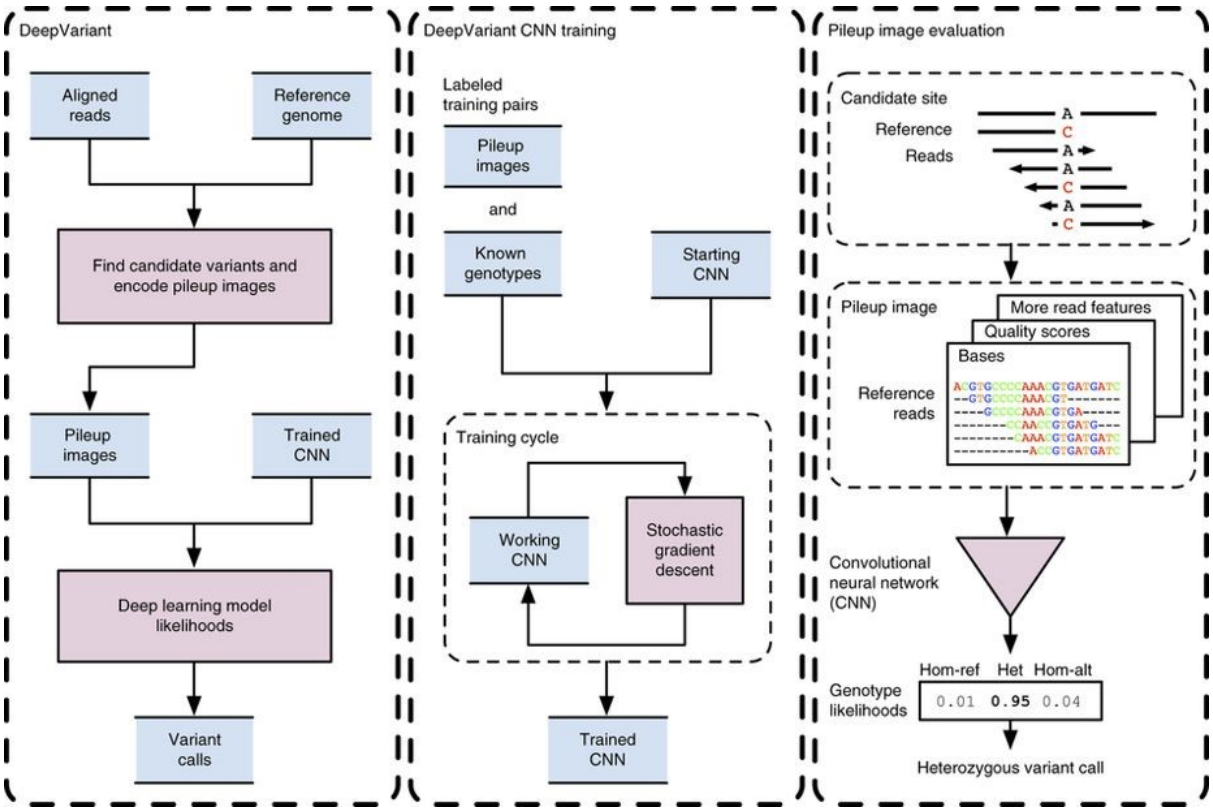


Figure 6. DeepVariant's inference and training workflows. Reprinted with permission from Poplin et al.[76].

GARFIELD-NGS

Architecture. GARFIELD-NGS utilizes four MLP models with five hidden layers each. The models are intended for different types of data and variants (an indel and an SNV models for the callsets from Illumina and Ion Torrent sequencing data) and are bound to the appropriate hyperparameters. For instance, both indel models are activated with ReLU, while SNV models are activated with Tanh; the layers' dimensions and regularization (L1 and L2) values were determined as the best-performing among randomly assigned during hyperparameter optimization. For the parameter optimization of the models, the network uses ADADELTA algorithm, a version of the SGD algorithm with an adaptive learning rate; its hyperparameters epsilon and rho were drawn out of the random search, too.

Training. The training data consisted of variant callsets with both true and false variants, identified. The sequencing of the HG001 sample have been conducted internally[79]. To train each of the four models, the data was split to pre-training, training, validation and testing sets. An autoencoder have been applied to the pre-training sets, and the weights of every prediction model were initialized from the pre-training ones. The search for hyperparameters that was decisive in building of the prediction models, was conducted with a 10-fold cross-validation. The resulted models were evaluated with the test sets, containing 50% of variants in the initial

sets. Lastly, the models were validated using additional WES callsets, that have not been part of the initial data.

I/O. The models employ 18 variant annotations as features for Ion Torrent variants and 10 annotations for Illumina variants. For each analyzed variants, the network outputs a 'confidence probability'.

Rationale. Instead of developing another variant caller, the authors aim to refine variant callsets produced by the widely used GATK HaplotypeCaller v3.6 (for Illumina data) and TVC v5.0.2 (for Ion Torrent data). Both of these callers produce sets of variant annotations associated with each variant, albeit different. The choice of the architecture to base models on was dictated by the framework (H2O.ai[95]), where the only natively supported neural network is the MLP.

VariationAnalysis

Architecture. FFNN of the tool v1.2, described in the preprint by Campagne et al.[77], comprises four fully connected hidden layers, all activated with ReLU. In v1.2.2 (not described in the preprint), the LSTM layer(s) were introduced (one by default) to work with indel sequences, along with the two other fully connected layers. The number of both kinds of layers can be configured with the tool's arguments. The LSTM layer(s) are activated with the Softsign (defined as $f(x) = \frac{x}{1+|x|}$, similar to the hyperbolic tangent). The model is initialized with the Xavier algorithm. The network uses L2 regularization and an AdaGrad optimizer, another variation of the SGD algorithm. The latest version of VariationAnalysis is v1.4.0, retaining the architectural details of v1.2.2.

Training. The network was trained on the NA12877 sample and truth data from Platinum Genomes (Chapter 5)[90]. The data have been divided into the training, validation and testing sets with a 80/10/10 proportion. The validation set is used to incite early stopping of training when the performance does not improve for 10 epochs.

I/O. VariationAnalysis utilizes a number of 'feature mapping' functions that convert the input alignment around a variant site into a tensor with selected features, such as number of variants in read, mapping quality, base qualities, strand orientation. The features are 'label mapped' to a heterozygous or homozygous genotype label. The output layers define the identity of the genotype at each variant site, whether it a base mismatch (A, C, G, T, N) or an indel, and their probability of being called.

Rationale. The authors aimed to build a flexible and efficient model that does not require heavy preprocessing such as conversion of the alignment to images (tensors) as in DeepVariant. The 'feature mapping' functionality allows to select additional or reduce unwanted features, experimenting with accuracy of prediction. The authors claim that the resulted architecture is able to work with polyploid

organisms (e.g. with plants) and different sequencing instruments, in addition to their previously developed somatic model (that is not discussed in this work).

Summary table

Table 1. Summary table, showing architectural details of networks of every tool.

	Clairvoyante	CNNScoreVariants	DeepVariant	GARFIELD-NGS	VariationAnalysis
Architecture	CNN	CNN	CNN	MLP	FFNN / FFNN+LSTM
N of hidden layers	5	7	9 enclosed	5	4 / 6+1
Initialization	He	Glorot	Pre-training network	Pre-training network	Xavier
Regularization	L2, dropout	Dropout	Auxiliary classifiers	L1, L2	L2
Activation	SELU	Linear / ReLU	ReLU	Tahn / ReLU	ReLU / Softsign
Optimization	SGD (Adam)	SGD (Adam)	SGD	SGD (ADADELTA)	SGD (AdaGrad)
N of parameters	1631496	1681156	Unknown, probably between 6797700 and 25000000 (GoogLeNet and Inception v2)	Unknown, allegedly between 5600 and 18600 depending on the exact model	Unknown

Chapter 5: Workflow and data management

This chapter explains the study design and the practicalities of the project. It starts with the general workflow, then elaborates on the materials used. This is followed by the variant calling workflow that was introduced in Chapter 2, here supplied with the particular choices and justifications. In the end, the means of the evaluation of the results are detailed.

The overview of the project

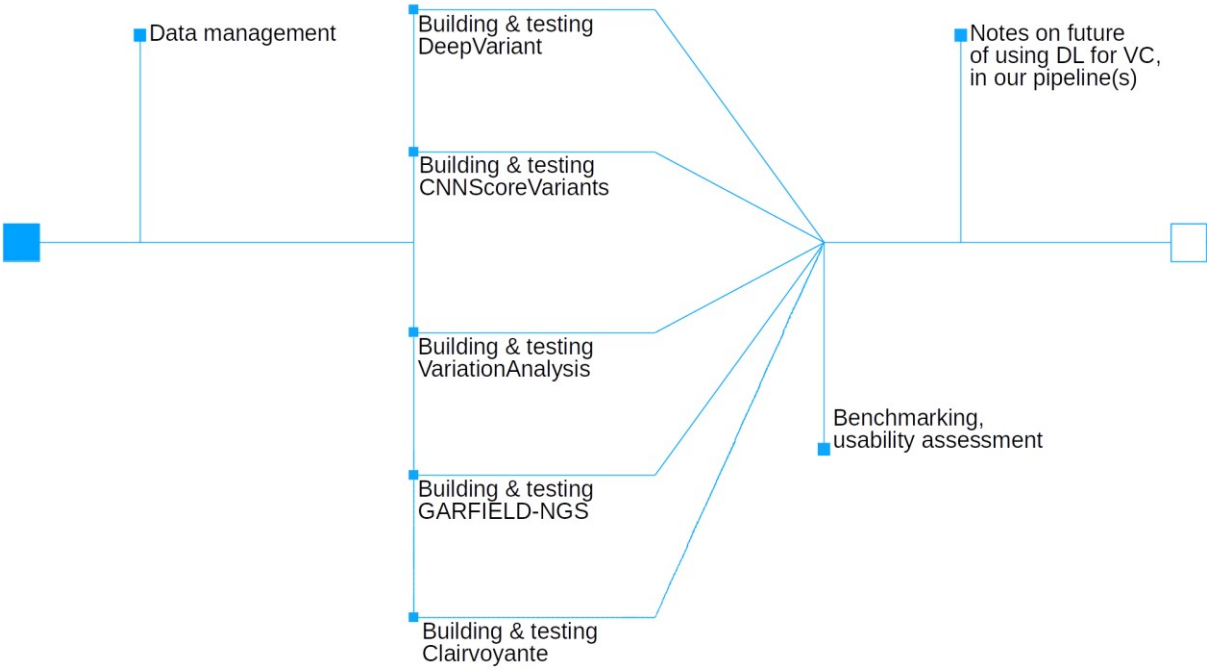


Figure 7. The flowchart depicting the progression of the project.

The practical side of the project has been carried out in several steps (Figure 7). First, it was necessary to account for and prepare the computing environment, as the hardware and all possible virtual environments and software packages' features and limitations have a significant influence on the depth of the study that can be performed in the field. Second, the data flow for the future pipelines had to be determined, including the kind of files to be processed, the kind of files to be expected in the end, as well as the order of processing. Third, the pipelines in question had to be designed and built, involving a selection of the particular tools and settings for the pre-processing and processing steps. Finally, the pipelines had to be thoroughly evaluated in terms of their performance and fitness to the existing genomic workflows.

Setting up the computing environment

Cluster

As it was noted in previous chapters, genomic and specifically alignment and variant calling applications rely on statistical models, mathematical optimization, and pattern recognition. While this alone does not have to be computationally demanding, genomic analysis is normally large-scale and as such, it benefits from the extensive use of computational resources. As an example, a recommended amount of WES samples for joint genotyping with GATK is no less than 30, with any of them taking up to 10 GB of disk space[96]. Further, WGS data files can easily occupy 50-500 GB per sample. Researchers' workstations are seldom designed to handle that kind of analysis once, let alone do it efficiently and on a regular basis.

At the moment, the bottleneck is resolved by two methods, often in combination. First, the problems are decomposed and operations are parallelized. This way, CPU threads, cores or computing nodes can be made to perform their part of the job as distinct tasks, effectively scaling computational capacity for the right kind of problem. Second, as with many other heavy operations, a high-performance computing (HPC) infrastructure - a server, a cluster of servers, a datacenter - is utilized. Some problems and tasks remain aggregated, but may still require large amount of RAM or a dedicated CPU (vCPU) available.

The cluster environment, however, does have certain restrictions. Because it is shared by many users, the system administration needs to balance their needs with the reliability and availability of the system. This implies that software packages or their dependencies that are not readily available in the official, supported repositories might not be installed or updated properly. If that is the case, it might be more feasible to run the required software in a Docker[97] container (a self-contained virtual environment) or in a Conda[98] virtual environment.

In addition, the computational resources on the cluster are allocated via the job scheduler. The scheduler is accessed with a range of batch commands that can put the jobs on a queue and request information about their behavior (logs and parameters such as states, memory usage, time of execution). The scheduler is immensely useful for the long-term projects, because it allows to monitor the pipelines.

Cloud with GPU

An investigation of multiple deep learning techniques, especially with stated amounts of data, automatically implies the utilization of a considerable amount of computational resources. The cluster was initially considered sufficient, however, the testing showed that the processing of the large WGS files leaves room for improvement (see Chapter 6). Since deep learning training and inference are working well with parallel processing, and since GPUs (graphics processing units)

are designed for the task, it was decided to get access to the GPU in the cloud. Fortunately, local centre for scientific computing provides that kind of resource for research institutions.

The GPU comes as a part of a virtual machine (VM) that is, in turn, based on an actual compute node. Three flavors, or types of VM with graphics processors are available: with 1 GPU, 2 GPUs and 4 GPUs. Each GPU is accompanied by 14 cores of vCPU and 120 GiB of RAM. None of the deep learning tools, at the time of the writing, is able to utilize more than 1 GPU for the inference. In theory, it is possible to simultaneously process different methods or parameters or samples on different GPUs. In practice, it is out of the scope of this work. Therefore, a 1-GPU node had been selected for testing.

The only available GPU to be used in a GPU node was NVIDIA Tesla P100, a high-end card with 16 GB HBM2 memory, 3584 CUDA Cores, and an official support of FP16 and FP64 (half-precision and double-precision floating-point, respectively) computation[99]. This makes it one of the top performing graphics processors for both deep learning and pure numeric operations such as scientific simulations. Therefore, it should be noted that with any other GPU the run times and even the capacity to run heavy models may vary.

The VM instance is running on the OS of user's choice, in this case it is Ubuntu 16.04 because of the support from NVIDIA and a large user base. Inside the VM instance, that is accessed via a secure shell protocol, the user must install version-specific NVIDIA CUDA (name of the parallel computing and graphics API for the general-purpose programming tasks[100]) Toolkit and corresponding libraries, such as cuDNN (Deep Neural Network library), cuBLAS (for basic linear algebra), cuSOLVER (for accelerated calculation of selected applications), cuRAND (for generating random numbers), cuFFT (for accelerated Fast Fourier Transforms), and CUDART (for CUDA runtime API)[101]. It was important to install the version that is compatible with frameworks that were used to train the neural networks. For the convenience of running the tools, the NVIDIA-Docker[102] and Conda have been be installed as well. The root access for the instance was provided by default, so the system update did not pose a challenge.

Data management

Input / output

In general, processing of the resequencing data requires at least 2 types of inputs. Those are the sequences to be analyzed ('raw' sequences that come in the FASTA or FASTQ format) and a reference genome to map against (next section). For the position-based variant calling, one also needs to obtain the targeted regions file (comes in the BED format) and, if needed, fulfill filter-specific requirements. The latter can differ: for instance, VQSR toolset of the GATK package needs known

variants sets to learn to filter out false positive (FP) variants, while CNNScoreVariants from the same package relies on the pre-trained model to do the same job.

The roots of the 'raw' sequences format are in the sequence similarity search software FASTP, developed in 1985 by W. Pearson and D. Lipman[103,104]. Today, the file format used by that software and its successor FASTQ (that includes quality metrics) are the *de facto* formats for storing nucleotide and peptide sequences[105]. The quality scores are defined by the sequencer software and represent the probability that the wrong base have been called:

$$Q_{phred} = -10\, log_{10}(e),$$

where Q scales exponentially, as in $Q = 10$ means 10% probability of the base being wrong base, $Q = 20$ means 1% probability, and $Q = 30$ means 0.1% probability, and so on. The score rarely exceeds 40. The qualities are expressed in a range of ASCII symbols from 33 to 126:

```
! " # $ % & \' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B
C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \\ ] ^ _ ` a b c d
e f g h i j k l m n o p q r s t u v w x y z { | } ~
```

The following represents what an individual read stored in FASTQ file may look like:

```
@HWI-D00360:6:H81VLADXX:1:1101:1245:2105 1:N:0:CGATGT

TTTTTTTCTGAGGCAAGTCCCACTCTCTTGCCCAGGCTGGAGTGCAGTAGTGTGACCTCGGCTCACTG
CAACCTCCGTCCCCCAGGTTCAAGTGATTCTCCTGCCTTANCCTCCCAAGTNNCTGNGNTCACAGGNN
NCCACCATCATG

+

@@@DDDDDD?DBFFF?GCFGEHIE@GGIIEGGIICBFEC<CGG;=B<FCDCDFE@DG:DA<EEEEE;(
;>CCBCCB:9<BBBB9?7::ACCCD>@DDDCCC@@@BCC@#++<@BBCC<C##++8#+#+++8<C1##
#+++8@B@CAC:
```

where the first line is an identifier, containing the instrument name, run and flow cell ID, lane, tile, instrument's camera pixel coordinates, read number, filter flag, and sample name; the second line is the read sequence; the third line is an actual plus sign; the fourth line consists of the phred quality scores.

In the example above, the quality scores in a more human-readable form would look like the following:

```
31, 31, 31, 35, 35, 35, 35, 35, 35, 30, 35, 33, 37, 37, 37, 30, 38,
34, 37, 38, 36, 39, 40, 36, 31, 38, 38, 40, 40, 36, 38, 38, 40, 40,
34, 33, 37, 36, 34, 27, 34, 38, 38, 26, 28, 33, 27, 37, 34, 35, 34,
35, 37, 36, 31, 35, 38, 25, 35, 32, 27, 36, 36, 36, 36, 36, 26, 7,
26, 29, 34, 34, 33, 34, 34, 33, 25, 24, 27, 33, 33, 33, 33, 24, 30,
22, 25, 25, 32, 34, 34, 34, 35, 29, 31, 35, 35, 35, 34, 34, 34, 31,
31, 31, 33, 34, 34, 31, 2, 10, 10, 27, 31, 33, 33, 34, 34, 27, 34,
2, 2, 10, 10, 23, 2, 10, 2, 10, 10, 10, 23, 27, 34, 16, 2, 2, 2, 10,
10, 10, 23, 31, 33, 31, 34, 32, 34, 25
```

Clearly, most (127 out of 148) bases have quality score higher or equal than 20, and 103 out of 148 higher or equal than 30. This is not the best possible read, although its source has more than sufficient average quality metrics. It is also apparent that the read sequence quality starts to deteriorate closer to the end - which is expected.

Aligned sequences are commonly stored in the BAM (BGZF-compressed SAM, a 'sequence alignment map' text-based, tab-delimited file) format, describing the alignments read by read, preceded by the header section[106,107]. The following example is an excerpt from the header of the BAM file corresponding to the FASTQ sequence shown above:

```
@HD   VN:1.5  SO:coordinate
@SQ   SN:1  LN:249250621
...
@SQ   SN:22   LN:51304566
@SQ   SN:X  LN:155270560
@SQ   SN:Y  LN:59373566
@SQ   SN:MT   LN:16569
...
@RG   ID:HWI-D00360:6:H81VLADXX:1:1101:1245:2105
      LB:LIB-HG001-NA12878
      PL:ILLUMINA
      SM:HG001-NA12878
      PU:H81VLADXX.1.CGATGT
@PG   ID:bwa
      PN:bwa
      VN:0.7.17-r1188
      CL:/app/bwa-0.7.17/bwa mem -t 16 -M /data/reference-
genome/hs37d5/bwa0717/hs37d5 /data/precision/HG001-NA12878-
50x_1.fastq.gz /data/precision/HG001-NA12878-50x_2.fastq.gz
```

where @HD line represents the first line of the header, containing VN (version number) and SO (sorting order); @SQ lines represent reference sequence dictionary, with contig (most often chromosome) names and their lengths; @RG represents read group, with the identifier, library, sequencing platform, sample name, and platform unit barcode; and @PG represents the program(s) used for the alignment and quality control.

Read lines in the alignment section contain, column by column: template name, segment flags, sequence (contig) name, position of the given aligned sequence segment, its mapping quality, CIGAR string, the next sequence (contig) name, the next aligned segment position, template length, the segment sequence itself, and its quality scores. In addition, the read line can be extended with 'optional' fields, carrying information such as edit distance to the reference, software-specific alignment score, or duplicating data from the header. In this example, mandatory fields for one read line are shown, separated by newlines for clarity:

```
HWI-D00360:6:H81VLADXX:2:2103:5668:31370
65
1
9998
19
48M1I63M1I35M
8
62666624
0
CCATAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCCAAACCCTACCCCTAAC
CCTAACCCTAACCCTAACCCTACCCCTACCCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCC
TAACCCTAACCC
DDGHGKGHGIHKGGGIHKGGGJILHHHJILHHHJILHHHIIJGHHJILHHH;,E&DEC.DDEHE:DFE
FFGEFEGDG?<FEEB>4AFE<@*<FD=@@EGDEC@FFDHD'AA<E@'<DDED>AFDCDFFFDEE>BEF
08(+=><@(A?E
```

The BAM files can be indexed to allow random retrieval of the alignment. The indexes are stored in the BAI binary files, consisting of BGZF block offsets.

Variant callsets are commonly stored in the VCF format, developed by the Global Alliance for Genomics and Health (GA4GH). Text-based files also exhibit a certain structure: they start with a header with meta-information lines (describing formats, filters, contigs, and software-dependent info) that are followed by a data table with variants and their properties. For instance, the header may look like:

```
##fileformat=VCFv4.2
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Approximate read
depth ...">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype
Quality">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
...
##INFO=<ID=AC,Number=A,Type=Integer,Description="Allele count in
genotypes ...">
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency
...">
##INFO=<ID=BaseQRankSum,Number=1,Type=Float,Description="Z-score
from Wilcoxon rank sum test ...">
##INFO=<ID=SOR,Number=1,Type=Float,Description="Symmetric Odds Ratio
...">
...
##contig=<ID=1,length=249250621>
...
##contig=<ID=22,length=51304566>
##contig=<ID=X,length=155270560>
##contig=<ID=Y,length=59373566>
##contig=<ID=MT,length=16569>
```

While the body with variants table may look like:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	HG001-NA12878
1	10616	.	CCGCCGTTGCAAAGGCGCGCCG	C	263.73	.	AC=1;AF=0.500;AN=2; BaseQRankSum=4.101; DP=24;ExcessHet=3.0103;FS=8.820; MLEAC=1;MLEAF=0.500; MQ=45.70;MQRankSum=0.419; QD=15.51;SOR=0.048	GT:AD:DP:GQ:PL	0/1:9,8:17:99:301,0,493
1	12198	.	G	C	60.28	.	AC=2;AF=1.00;AN=2; DP=3;ExcessHet=3.0103;FS=0.000; MLEAC=2;MLEAF=1.00; MQ=24.73;QD=20.09;SOR=2.833	GT:AD:DP:GQ:PL	1/1:0,3:3:9:88,9,0

where the #CHROM column represents current contig; ID column represents possible variant identifier from the dbSNP database; REF represents base(s) from the reference genome and ALT - from the investigated (alternate) one; QUAL column is for phred quality scores; FILTER values are added by the filtering methods and can be either ‘PASS’, a list of failed filters, or missing; INFO fields contain various information about a particular variant/base, that can be helpful for filters (variant annotation); finally, FORMAT and SAMPLE columns indicate genotype status, likelihoods and qualities.

It should be noted that certain variants may be recorded in more ways than one. For instance: repetitive insertions may be represented as several smaller or one larger insertion (‘GCAT → GATCACAT’); a deletion in a region of identical bases (homopolymer, ‘TAAA → TAA’) may be represented as a deletion of either one of these bases; complex variants may or may not be described as conjoined indels and mismatches, only indels, or only mismatches (‘TAAA → TAT’).

FASTQ, FASTA, SAM and VCF files can be compressed; as a convention, it is done in BGZF (Blocked GNU Zip Format)[106]. The blocked compression allows faster random and selective access, while the compression ratio remains almost as high as with GNU Gzip (shown as size reduction percentage in the output of the *gzip -l [file.gz]* command, from 73.08% to 74.80% for GNU Gzip and from 72.86% to 74.26% for BGZF, assessing the WES raw data files discussed in the 'Sequencing data' section). Some of the sequencing data processing tools are able to decompress compressed FASTQ and FASTA files with a built-in functionality, while others may require a decompressed input. The BGZF compression or decompression can be performed with e.g. the *bgzip* tool from the *HTSlib* toolset[107,108].

Reference genome

The reference genome holds the genetic information of several different individuals. Being an assembly by design, it is sub-optimal and contains gaps, variations, and quality issues. Therefore it is regularly updated with the latest reliable findings and techniques. The chosen reference, however, is not always up-to-date, as it is bound to the target library preparation kit (from the manufacturer of sequencing equipment) that was used for the enrichment, and a corresponding targeted regions ("manifest") file with genomic coordinates from a particular reference genome.

The reference genomes are assembled and curated by Genome Reference Consortium (GRC). Currently, there are two major versions of the human reference genome that are in production: GRCh37 (b37) and GRCh38 (b38). Both of them exist in several alternate editions that may or may not include ALT contigs, non-chromosomal contigs (mitochondrial and unlocalized (GL*) sequences), not inherited additions such as viral genomes, as well as updates over the previous builds. In addition, the builds may follow a different segment nomenclature, such as '1' in GRCh37 or 'chr1' in hg19 (an initial version of GRCh37 released by UCSC, containing only the original contigs), which can be converted with a 'liftover' procedure[109,110]. The reference genomes are supplied and stored in the FASTA file format.

Here. the reference genome used is hs37d5. It is composed of the chromosome sequences, mitochondrial sequence, HPV (Human herpesvirus 4 type 1) sequence, GL* contigs, and a decoy sequence[111,112].

Sequencing data

The selection of the test dataset was based on the following variables:

- ➔ Public availability: the data should be accessible to allow testing and evaluation procedures to be reproducible.
- ➔ Availability of the truth set: the evaluation involves a comparison of the variants called using the methods under evaluation to the actual variants.

- ➔ Sample type and size: WGS, WES or targeted samples require allocation of different amount of resources (the less the better) and can be more or less practicable for a given type of a study (individual, cohort or population; rare or common variants seeked).
- ➔ Sample source diversity: relates to the type of a study; in this case, several individuals are more desirable because the goal is to test how the new methods perform with the different setting and data.
- ➔ Sample bias: involvement in any training of any of the tools in the comparison supposedly renders the sample unsuitable; the comparison of tools' performance on the biased vs unbiased samples, however, may present an additional interest.
- ➔ Sequencing technologies: alignment and variant calling tools' documentation often informs of the specific method the tool works best with; in addition, sources of research and clinical genomic data often utilize methods that are more convenient or conventional than others.

The sample sources included the Utah family with the pilot genome NA12878 (HG001) collected by Centre d'Etude du Polymorphisme Humain (CEPH), and 2 trios from the Personal Genome Project (PGP) they have participated in[79]. The trios are Ashkenazi Jewish family of NA24385 (HG002, son), NA24143 (HG003, father) and NA24149 (HG004, mother), and the Han Chinese family of NA24631 (HG005, son), NA24694 (HG006, father) and NA24695 (HG007, mother). Their genetic material (cells and DNA) is distributed by the Coriell Institute for Medical Research, NIST (National Institute of Standards and Technology) and several companies for both non-commercial and commercial purposes. For example, NA12878 may serve as a source of the control samples in routine sequencing experiments. In fact, this material have been extensively sequenced over the last several years. As a result, a number of different sequencing methods have been used to obtain the genetic data of listed individuals.

Sources of the data that have been considered include all Genome In A Bottle (GIAB) samples[79], precisionFDA Truth Challenge samples[93], and Illumina Platinum Genomes samples[90]. Genome in a Bottle is a name of a NIST-based consortium that aims to characterize and standardize human genomic data for the research and clinical use. The consortium collects and studies the sequencing data of NIST Reference Materials genomes mentioned above. The relatively regularly updated variant datasets are considered to be among the few high-confidence variant callsets ('ground truths') in the industry. The GIAB has also contributed WGS data of the NA12878 and NA24385 to the precisionFDA Truth Challenge, an open contest organized by the FDA to improve the consistency and accuracy of variant calling pipelines. In fact, the chemical and sequencing parameters (sample preparation kit, read length, target insert size, instrument) of one particular batch of the samples at GIAB (HiSeq 300x) is remarkably similar to that of the precisionFDA samples. In addition, the NA24385 sample from the challenge had been adapted for the 'case

studies' (testing and demonstration) of one of the tools evaluated in this work (DeepVariant). All of these sources are summarized in Supplementary table 1.

Platinum Genomes sample data have been generated exclusively by Illumina, the sequencing equipment manufacturer. Unlike the trios, this dataset consists of 17 family members, i.e. father and mother and their parents and a total of 11 children. This pedigree allowed Illumina to explore the range of familial variation, i.e. infer haplotype inheritance and genotype combinations.

In the end, the primary sequencing data for this project was selected to be from Oslo University Hospital - as the only exome sample set that spans across four individuals and that did not participate in training of most of the tools (with the exception of DeepVariant models that were partially trained on HG005/NA24385), and are compatible with the truth set from GIAB, and were sequenced with the one of the most widespread platforms (Illumina HiSeq 2500). For the variant calling evaluation, the data was pre-processed and downsampled to the 10% of the original coverage with a 10% step (Supplementary table 2). Such a setup allows to estimate the accuracy of variant calling from the real data that, for economical or technical reasons, may be well below the recommended coverage. The secondary sequencing data was selected to be from the precisionFDA Truth Challenge - as the known genome sample set that had been utilized in a large variant calling competition and that is also compatible with the truth set from GIAB, and that contains one biased sample (HG001/NA12878).

The data files of the exome GIAB samples are available on the FTP site of NCBI (National Center for Biotechnology Information)[113]. The original sequencing data for them is available by the SRA accession numbers given in the data description publication. The precisionFDA dataset can be downloaded after registration at the precisionFDA portal, and the adapted versions are available in Google Cloud buckets. The Platinum Genomes can be downloaded from the Illumina FTP site, as well as Illumina, Amazon AWS, and Google Cloud repositories[114,115].

Variant datasets

Variant datasets of various kinds are used to train algorithms and validate their work. These resources may differ in the degree of confidence (in the authentic variant existing at a particular position), the source of original sequencing data, and the known state of the included variants (the presence in the variant database, as of the time of writing).

Gold standard, also called ground truth and high-confidence variant callsets, are rare: as of this writing, only three exist for short variants (GIAB, Platinum Genomes and the new CHM-eval from SynDip)[80,88,89,90]. They are always defined in the confidence intervals; the GIAB variant callsets, used in this work, come with BED files that are later used in the evaluation - for the purpose of not wasting computational resources on the whole reference genome. Their development

consisted of integrating and curating several high-quality datasets from different sequencing instruments and variant callers; the development continues[62]. The version of all GIAB truths in this work is 3.3.2.

Individual variant's biological and clinical consequences may be recorded. In dbSNP database[4], SNVs (SNPs) and small indels are assigned an accession number in 'rsID' format. Variant callers and callset evaluation tools, as well as human experts can use it to access associated information (position, gene, consequence, frequency, publications, possible phenotype). Not many dbSNP variants, however, have a known clinical outcome; such variants are collected in the OMIM and ClinVar databases[116,46], while 'normal' or common variants are covered by the HapMap[117] and 1000 Genomes[118] databases.

Setup and usage of the pipelines

Given that the analysis is supposed to be conducted regularly, the researcher should be able to use the tools with as little hassle and as much reproducibility as possible. This imposes certain features had to be built in the pipeline, such as comprehensible order of execution, management of I/O paths, and the ability to distribute across multiple files or parallelize across multiple threads where necessary. In addition, in order to achieve interchangeability of the variant calling pipelines that is necessary for this project, a common ground had to be established. The pre-processing procedure is well-suited for this purpose, as the majority of variant callers use mapped sequences in BAM file(s) as an input. The GATK Best Practices have been used as a blueprint to create a pre-processing pipeline[41,59].

Pre-processing

During pre-processing, 'raw' reads from the sequencer get ready for the variant calling. The data passes through the following phases:

1. Quality Control (QC), the process of collecting quality metrics of reads and adjusting however necessary. QC is performed before and after alignment. It starts with locating and trimming the reads with bad base qualities or contaminated with adapter sequences (as of today, a rare occurrence in DNA sequences). For quality assessment of the FASTQ files, often a combination of tools such as FastQC, HTSeq, Kraken, multiQC is applied. For trimming, tools such as Trimmomatic, Sickle, BBDuk or others may be applied. After alignment, positions and structure information for each read become available, allowing for the new quality scores. Based on this information, the Broad Institute's Picard and GATK toolsets offer the way to assign read groups (if not done or incorrectly done during alignment), merge multiplexed reads (from multi-sample lanes), and detect and remove (or correct) library preparation or sequencing errors, reducing potential bias and consequent false positive calls.

Here, QC included:

a. no trimming as FastQC demonstrated ‘good’ quality of all data files[119];

b. marking duplicate reads using MarkDuplicates tool from the Picard toolset, that inputs a sorted alignment SAM/BAM file, sifts through the records and collects read pairs with 5’ end coordinates (locating them via SAM flags), traverses and ranks reads in 'duplicate sets' (containing reads that are comparable by barcodes, coordinates, orientation) based on quality score, and adds the duplication flags to all but the highest-ranking ('representative') read records in the output BAM[120];

c. recalibrating (calculating and comparing) base quality scores using BQSR method (BaseRecalibrator + ApplyBQSR + AnalyzeCovariates + PrintReads tools) from the GATK toolset, that finds sequence mismatches and adjusts observations[41];

d. collecting alignment, exon- and genome-specific measurements using CollectAlignmentMetrics, CollectHsMetrics, CollectWgsMetrics respectively from the Picard toolset.

2. Alignment, the process of finding and adding position, quality, structure information for each read, complemented by sorting the reads according to the position. The given sequences, genome or exome, are compared against the reference genome, and mapped according to the discovered matches. The goal of the mapping is to find the correct position of each sequence, considering possible sequencing errors and actual differences in the genetic code (variations).

   With the amount of data produced by high-throughput sequencing experiments, the alignment requires effective computation methods. Hence, a number of algorithms specifically for the alignment of short reads have been developed. Most of these algorithms are index-assisted and belong to the family of dynamic programming methods. The problem they are designed to solve is called ‘approximate matching’ problem, that is based on the idea of defining the distance (edit distance) between one string and another. At first, the indexing algorithm sifts through the available genome and compiles an index, or a binary dictionary, with sub-strings or suffixes of particular patterns, either sorted or grouped. This index allows to perform fast ‘exact matching’ for sequences (strings) that share identical parts with the query, ridding the dynamic programming approach of the regions in genome that don’t have a match with the particular string. The index, however, does not operate with mismatches and gaps (substitutions and indels) that naturally occur in a library of short reads.

The dynamic programming complements the index by looking for the 'approximate matches' in the indexed regions (of the reference genome). There, the strings are compared against other strings on the dynamic programming matrix, with reads' bases represented as rows and selected regions' bases represented as columns. The matrix is then filled with edit distances between the row and the column bases, and with regard to the penalties that correspond to mismatches and gaps in the alignment. After that, the matches can be traced back to their locations.

In biological sequence analysis, the 'approximate matching' problem can be broadly divided into 'global alignment' and 'local alignment' problems. The goal of global alignment is to identify occurrences of reads (sequences) along the reference genome; the goal of local alignment is to identify most similar parts of a pair of sequences.

An appropriate selection of a method or software, as mentioned in Chapter 2, depends on the data type (DNA, RNA, ChIP-Seq, other), sequencing platform (some methods take into account specific technological characteristics like quality degradation in Illumina, color coding in SOLiD, or homopolymer segments reading in Ion Torrent instruments), computational capabilities, expected performance and recommendations from the developers of downstream analysis methods.

Here, alignment was performed by the *BWA* tool using the BWA-MEM algorithm[121] with -M argument (resolving potential MarkDuplicates issue of not understanding splitting of the alignment that often happens in local alignment), along with the sorting by *samtools*[107,108].

Processing

The tools are fed inputs according to their documentation. For the tools that are designed to process alignments and output variant callsets, the alignment files of the mentioned sequencing data are given as input. For the variant filtering tools, the input is a 'raw' callset from the GATK HaplotypeCaller, that was made to process the same alignment files.

Evaluation

The computational load was measured in CPU-hours (time spend as computed) and in wall time hours (time spend as perceived). The two can be different due to the extensive parallelization of processes. The final part of the assessment described how well does a particular variant caller fits into the production pipeline, taking into account the effort it required for the set up and how user-friendly it is in everyday use.

The performance evaluation part of the project included the variant calling benchmark, and the assessment of the computational load, flexibility and ease of use. The variant calling benchmark was performed according to the established best practices.

Fitness evaluation

Convenience in use

As a part of the pipeline, the tools must be able to process multiple samples repeatedly or at the same time. It depends on tools' ability to be executed via a SLURM command or script (starting multiple instances, feeding multiple inputs).

Required software dependencies may or may not be difficult to install globally, locally or virtually. For instance, they might conflict with other versions already installed or be challenging to run coordinatively if the computing or scheduling platform imposes restrictions. Creating a local environment implies additional obstacles in updating to a newer version of the software.

Computational efficiency

For algorithmic problems, the computational cost is commonly approached as computational complexity. For algorithms, the complexity is commonly defined by a 'big O notation': $O(n)$, where n represents the amount of steps an algorithm takes for its execution[122,123], such as n,n^2, 2^n, $log(n)$. In machine learning and especially in deep learning, the models contain a multitude of steps (iterations and inner steps) that may or may not be solved in Polynomial time by the optimization algorithms[124]. A model converges when it reaches its desired accuracy or mathematical optimum, which generally depends on the amount of parameters and may take considerable time. As for any algorithm, this is considered a practical limitation, and techniques for reducing the number of parameters by e.g. compression of filters are actively developed.[125-128] This number, however, is not always explicitly stated or can be derived from the documentation (Chapter 4).

For a software or a pipeline executed on the cluster, the amount of resources utilized for processing influences whether the particular software is used instead of the competitor. The resources are seen as the wall (perceived) and computing time spent, and the consumption of disk storage and memory. Such information can be extracted from the scheduler or system utilities such as *time*, *du* and *df*.

Performance benchmark

The general guidelines, metrics, and the toolset for benchmarking small variants are developed and provided by the Global Alliance for Genomics and Health (GA4GH), a worldwide organization managing the regulation and technical standardization of the usage of genomic data.

Per these recommendations, the datasets for the benchmark should include high-confidence (baseline, 'true') variant calls in corresponding confidence intervals, such as GIAB or Platinum Genomes. The test dataset from GIAB has been selected, as described in the 'Sequencing data' section. The comparison of the inferenced variant calls against the high-confidence ones can be accomplished by either

- a genotype match - 'replaying' (attempting to converge) query and truth haplotypes into each other and choosing the 'path' (a variation of the resulting sequence) with most TP and least FP and FN variants;
- an allele match - counting any agreeing alleles (including FP genotypes, as shown in Table 2) along the query and truth haplotypes;
- a local match - counting query variants that are within a matching distance of the truth variants (including FP genotypes and alleles, as shown in Table 2).

			Truth (high-confidence) variant set			
			reference	het_variant	het_variant	homo_variant
			ref/ref	ref/var1	var1/var2	var1/var1
Query variant set	**Negatives**					
	reference	ref/ref	n/a	FN	FN	FN
	Positives					
	het_variant	ref/var1	FP	TP	FP.GT	FP.GT
	het_diff_variant	ref/var2	n/a	FP.AL	FP.GT	FP.AL
	het_diff_variant	ref/var3	n/a	n/a	FP.AL	n/a
	het_diff_variant	var1/var2	FP	FP.GT	TP	FP.GT
	het_diff_variant	var1/var3	n/a	n/a	FP.GT	n/a
	het_diff_variant	var2/var3	n/a	FP.AL	FP.GT	FP.AL
	het_diff_variant	var3/var4	n/a	n/a	FP.AL	n/a
	homo_variant	var1/var1	FP	FP.GT	FP.GT	TP
	homo_diff_variant	var2/var2	n/a	FP.AL	FP.GT	FP.AL
	homo_diff_variant	var3/var3	n/a	n/a	FP.AL	n/a
	Other					
	no call (both alleles)	nc/nc	n/a	FN	FN	FN
	no call_reference	ref/nc	n/a	FN	FN	FN
	no call_variant	nc/var1	FP	TP	FP.GT	FP.GT
	no call_diff_variant	nc/var2	n/a	FP.AL	FP.GT	FP.AL
	no call_diff_variant	nc/var3	n/a	n/a	FP.AL	n/a

Table 2. Possible types of matches and mismatches of variants[6,129]. FP.GT = False Positive genotype variant, FP.AL = False Positive allele variant. Adapted with permission from Krusche et al., 2018.

Due to the possible differences in variant representation in query and truth VCF records, the use of the more sophisticated methods such as 'replay' may be desired for the more stringent matching. Alternatively, the local match method along with the manual curation would be more allowing for differences. The *hap.py* toolset[5], recommended by GA4GH, includes several variant comparison engines that support

these methods: *xcmp* (default, matching entire haplotypes or genotypes), *scmp* (local matching), *vcfeval* (a built-in part of the *RTG Tools* toolset, matching genotypes)[130].

The toolset calculates several metrics for the whole callsets and for parts of it that correspond to various conditions, taken as variant subtypes (indels of different lengths) and annotations. The metrics in use are defined as:

$$Precision = \frac{TP}{TP+FP}, Recall = \frac{TP}{TP+FN}, F1 = \frac{2 * Precision * Recall}{Precision + Recall},$$

where TP (True Positives) are variants that match with the truth set, FP (False Positives, Type I error) are variants that do not match with the truth set, and FN (False Negatives, Type II error) are true variants that are missed in the query. Note that TN (True Negative) would mean a match with a reference, or no variant at all; therefore, they are not present in the test callset from the GIAB, and obtaining specificity (true negative rate) values would not be possible[6,131].

The F1 score is defined as a 'harmonic' or balanced average of precision and recall, and is useful as a single measure combining them, and providing better discrimination. For example, the two different data points could yield different metrics, such as:

$Precision1 = 0.8, Recall1 = 0.7$, and $Precision2 = 0.9, Recall2 = 0.6$,

resulting in the same precision-recall average of both data points (0.75). The F1 score gives a better sense of their relative position: 0.747 for the first data point and 0.72 for the second.

The precision and recall metrics for variant conditions ('discrimination thresholds') are used to construct Precision-Recall curves. Different methods may call variants with more or less of these conditions (considered by the toolset), therefore the amount of data points generated by comparison with each callset varies. The plots in Chapter 6 were prepared with R, using the *happyR* developer package[132].

Versions

The following versions of the software have been used in this work.

For preprocessing: FastQC 0.11.7, Picard 2.17.8, BWA 0.7.17, BQSR of GATK 4.0.7.0, Samtools 1.9.

For processing: HaplotypeCaller, VQSR and CNNScoreVariants of GATK 4.0.7.0, Clairvoyante 1.0, DeepVariant 0.6.0 and 0.7.0, GARFIELD-NGS 1.02.

For evaluation: hap.py 0.3.10, R 3.5.1.

Chapter 6: Comparison of the variant calling pipelines

This chapter delivers the selection and the evaluation of the DNN-based variant detection methods. It starts with comparing their approaches to the problem, then reflects the usability and ease of integration with the existing genomic pipelines. Along with that, tools' use of computational resources is taken into account. Finally, the benchmark results are showcased.

Overview of the approaches

The neural network based variant callers (Clairvoyante, CNNScoreVariants (GATK), DeepVariant, GARFIELD-NGS, and VariationAnalysis) were designed to achieve the same goal (correct and accurate detection of the genetic variation), but their approaches differ. Neither of the presented tools is an all-in-one solution; all of them require pre-processing (which in this case is identical, as described in Chapter 5), some of them require post-processing (producing a subsequent analysis-ready VCF file). The DNN-based tools are supposed to replace or complement known, reliable solutions in the production-grade genomic pipeline. As such, the comparison includes these solutions, namely HaplotypeCaller producing 'raw' variant calls and HaplotypeCaller with VQSR-filtering from the GATK package; their working principles are detailed in Chapter 3. The deep learning -based tools are outlined in this section; their models are described in Chapter 4.

Clairvoyante

Implementation. Clairvoyante[73,133] is implemented as set of Python 3 scripts; it relies on the presence of GNU Parallel[134] utility in the execution environment and Python modules such as Tensorflow[135] framework for machine learning, intervaltree for handling eponymous data structures, and blosc for data compression. GNU Parallel allows concurrent computation, which is useful for heavy tasks like training and inference of neural networks, while Tensorflow provides multithreading; together, they can create multiple multithreaded 'jobs', each working on its own chunk of the data. A version of Tensorflow for running on the GPU, in this case, is only used for training. The execution can be sped up further by installing the optimized Pypy[136] version of Python. In the cluster, Python 3 and Pypy, and consequently Clairvoyante scripts were called through the Conda virtual environment.

Workflow. For input, the tool accepts an alignment file (BAM) along with a possible regions file (BED), reference genome, and arguments for the model and parallelization. The output is tens of regional callsets in separate VCF files. For the evaluation, as well as for a potential downstream analysis, they need to be concatenated into one VCF (done with *vcfcat*, part of the *vcflib*[137]) and then sorted (done with *vcfsort*, another part of the *vcflib*).

Behavior. Being the only variant caller with a CNN model initially developed for the analysis of long reads (Oxford Nanopore, Pacific Biosciences sequencing

instruments), Clairvoyante is a technical outlier in the reviewed group of variant calling software. However, the availability of the 'Illumina' model (though trained using the 'hg38' reference genome) allows Clairvoyante to participate in the comparison.

Once started, the tool extracts variant candidates with the help of *samtools mpileup* (Bayesian inference of genotype likelihoods)[107,138], creates tensors for each of the candidate variant sites, and inferences desired variants. In essence, it is a filtering approach, conceptually similar to the one implemented in CNNScoreVariants and DeepVariant.

CNNScoreVariants (GATK4)

Implementation. CNNScoreVariants[74,87] is included in the GATK4 toolkit as 'experimental' method (for the reviewed 4.0.7.0 version) for annotating 'raw' VCF files with scores. The main GATK4 engine requires installed Java environment for running; the tool, however, runs on Python and utilizes a custom package vqsr_cnn, which in turn relies on the following libraries: Keras[139] for deep learning (depending on Tensorflow and Theano[140] backends), scikit-learn[141] for machine learning, NumPy and SciPy[142] for mathematical and scientific computing, Matplotlib[143] for plotting, and Biopython[144], PyVCF[145] and Pysam[146] for processing biological data. Multithreading is supported; to utilize a GPU, a Tensorflow-GPU must be installed. Conveniently, GATK4 can be started from an official Docker image or a Conda virtual environment.

Workflow. CNNScoreVariants is operated from the GATK command line interface, with input in form of the 'raw' VCF file and a reference genome, and output in form of the revised VCF file:

```
path/to/gatk CNNScoreVariants
                            --variant [raw_variants.vcf]
                            --reference [reference.fa]
                            --output [revised_variants.vcf]
```

The output file is, in fact, almost identical to the input, with the addition of a new annotation 'CNN_1D' in the INFO column, denoting the score assigned by the neural network (the evaluation of the '2D' model was not conducted). After that, in order for the variants to be actually filtered (as was designed) and then evaluated, another GATK tool should be applied:

```
path/to/gatk FilterVariantTranches
                                    --variant [revised_variants.vcf]
                                    --resource [hapmap.snps.vcf]
                                    --resource [1000G.indels.vcf]
                                    --info-key [CNN_1D]
                                    --tranche [99.9]
                                    --tranche [99.0]
                                    --tranche [95]
                                    --output [filtered_variants.vcf]
```

Where the resources contain common variant sites, info-key indicates the new annotation, and tranche values designate truth sensitivity (recall) to them.

Behavior. The scoring tool traverses the input and feeds each previously called variant with its position, reference and alternative alleles, contextual window and several annotations, encoded into the tensors, to the selected model. The model's architecture and weights are loaded from their respective temporal files. Variants are processed in batches of variable size, and each variant is given a Log Odds Score.

The filtering tool traverses the variants twice. In the first pass, it seeks the variants with added score (the 'CNN_1D' value) and counts how many of these have corresponding sites in the resources. In the second pass, it compares the scores with the cutoff calculated using the counts of sites in resources, and filters out a number of filters that are below the cutoff.

GARFIELD-NGS

Implementation. GARFIELD-NGS[75,147] is a Perl script, reading and writing the data and invoking neural network models. The models are based on the H2O.ai[95] machine learning platform and enclosed into the Java packages (Jar). Thus, the tool can be executed in most of the *nix systems, especially ones targeted for HPC. Despite the theoretical possibility of engaging multiple CPU threads or a GPU, it does not happen and is not mentioned in the documentation; in fact, the models are small enough to allow both inference and training at a negligible cost.

Workflow. The script takes in a 'raw' VCF file (e.g. from HaplotypeCaller) and outputs another VCF file with added annotation '*sample-name_true*' in the INFO field, i.e.

```
AC=2;AF=1;AN=2;DP=2;ExcessHet=3.0103;FS=0;MLEAC=2;MLEAF=1;MQ=27;QD=1
8.37;SOR=2.303;sample-name_true=0.338
```

This file is to be filtered based on the recommended threshold (separately for Illumina and Ion Torrent data, for SNVs and indels). The authors do not clarify the preferred method of filtering. Two methods have been devised:

1. A Python script removing every variant surpassing the threshold. Proved to be cutting off too many variants, so the performance had suffered;
2. Four-step filtering using corresponding programs:
    a. Edit the annotation name to make it recognizable and filterable with

```
sed -i 's/sample-name_true/sample_name_true/g' input.vcf
```

    b. Separate indels and SNVs into separate VCF files with

```
vcftools --remove-indels --recode --recode-INFO-all --vcf
input.vcf --out output.snvs[.vcf]
```

and

```
vcftools --keep-only-indels --recode --recode-INFO-all --
vcf input.vcf --out output.indels[.vcf]
```

    c. Apply indel- and SNV-specific filter threshold with

```
rtg vcffilter -i [output.snvs.vcf / output.indels.vcf]
-o [output.snvs.filt.vcf / output.indels.filt.vcf]
--keep-expr 'SAMPLES.some(function(s)
{ return INFO.sample_name_true>REC_THRESHOLD })'
```

    d. Join indels and SNVs back together with

```
bcftools concat
output.snvs.filt.vcf output.indels.filt.vcf
-o output_joined.vcf -a -O z
```

The second method has been applied.

Behavior. The tested Illumina model works unexpectedly fast, spending only from a few seconds to a few minutes on one VCF file, depending on its size. The script reads the file and assesses the amount of SNV and indel variants; a multiallelic portion (up to ~1.5%) of them is dropped and therefore not analyzed. Two models are used sequentially: one for indels, another for SNVs. In the end, the predictions are written into the output VCF file.

DeepVariant

Implementation. DeepVariant[76,148] is written in Python and its main execution environment is Python; nonetheless, the reading and writing sequencing and serial data require specific versions of additional software packages like Protobuf and Nucleus, which in turn may need a specific version of a building system Bazel or other dependency[149]. The neural network relies on the Tensorflow framework. In addition, DeepVariant is hardware-dependent, as its binaries require support for certain vector instructions in the CPU (most x86-64 CPUs produced after 2011

support them). Altogether, it might cause compatibility issues in the cluster environment. This can be solved by preparing a local environment with all possible dependencies, or by running DeepVariant in a Docker image that the authors provide. The special version of a Docker image can utilize a GPU on an appropriate machine.

Workflow. Between the input BAM and the output VCF file, the tool passes through three steps: *make_examples* that converts alignments into tensor records, *call_variants* that performs the genotyping with the CNN, and *postprocess_variants* that converts the CNN output into a common format for variant callsets. The steps are implemented in the form of zip-packaged Python modules and therefore can be executed provided a Python interpreter and a correct set of arguments, i.e.

```
path/to/python path/to/deepvariant/exec args
```

The input and output options of the DeepVariant scripts allow to have it process only part of the data, thus making it easy to parallelize and scale using independent processes or even nodes. If necessary, the output of 'sharded' (parallelized) run can be combined into a single file. Although not explicitly stated in the documentation, the outputs of the first two steps are archived internal files, and as such they can be decompressed, concatenated and compressed again. Such an action might be useful if the plans or the available computational power change between the steps.

For the production environment, three scripts corresponding to these three steps have been developed. The scripts allow quick substitution of inputs and parallelization of the workload across multiple computing nodes in the SLURM scheduling system. The scripts are based on the 'case study' workflows from the authors of DeepVariant; they were adapted and extended to allow running with variable number of cores and running the second step (that involves CNN inference) either in the same way or on the GPU machine.

Behavior. The *make_examples* step looks for candidate variants from start to end of the supplied alignment. For convenience, it logs the amount of candidates per every 1000 positions. The *call_variants* step inferences variants in several hundreds of examples per batch, and logs it accordingly. The *postprocess_variants* step combines previously parallelized outputs if necessary, sorts variant calls according to the position, and writes them to the VCF file. For each variant, measures such as genotype likelihoods, qualities, and allelic depths and frequencies are logged.

VariationAnalysis

Implementation. The tool[76,150] is built upon a Goby framework[151] from the same Campagne Laboratory, which is intended to handle various aspects of NGS analysis, from DNA and RNA sequence alignment to somatic and germline variant calling, differential expression, methylation analysis, and various file conversions. The framework integrates multiple 'plugins' of corresponding methods, such as BWA for DNA sequence alignment or pieces of GATK for somatic and germline variant analysis. Goby reportedly has a web version called GobyWeb[152]; as of this writing, its demo is not available online, but the software is downloadable and can be installed on a web server. The installation of Goby requires the Java environment and the Apache Maven building system to be preinstalled.

Workflow. VariationAnalysis is used to train genotyping models that can be applied with Goby. During the training, it converts BAM alignments with the corresponding truth VCF into a proprietary Sequence Base Information (SBI) file format that contains features that are mapped to variant sites. The SBI file is then split into the training, validation and test sets using a supplied bash script, trained using another bash script, and tested in the same fashion. Alternatively, pre-trained models can be downloaded.

The prediction of the genotypes with a trained model is done by calling the Goby binary with arguments stating the input alignment, reference genome, format ('variant_discovery'), and a model.

Behavior. Unfortunately, running the scripts or the framework turned out to be unsuccessful. The described stages of the workflow, such as converting the alignment to the SBI format, training the model, making predictions with it, – often ended up with errors (about missing or unexpected command or option or file), without a clear possibility to troubleshoot. In addition, the documentation is not maintained well: part of it is outdated, part is missing.

Fitness evaluation

The presented tools must be able to complement or integrate with the existing variant calling pipeline. They exhibit two approaches: one is to filter the GATK HaplotypeCaller 'raw' output (CNNScoreVariants, GARFIELD-NGS, and the established method VQSR), and another is to control variant calling from the BAM alignments to the final VCF by the means of the tool under evaluation (Clairvoyante, DeepVariant, VariationAnalysis). Per Chapter 6, there is two parts to the fitness: the ability of being conveniently operated and the ability of being efficiently executed. The latter is demonstrated in a form of a summary table.

Convenience in use

Clairvoyante is dependent on Python 3 with additional modules and benefits from the use of Pypy and GNU Parallel, and therefore has to be executed via the environment

that supports at least Python 3 and the modules. In this case, Clairvoyante was run through the Conda environment system, installed for the user on the cluster, which had caused conflict with the SLURM scheduling system. When the tool was run from the shell, it did not generate other issues.

CNNScoreVariants, being a part of the GATK toolkit, utilizes a combination of Java and Python backends, with Java sending requests for processing to Python and expecting to receive an answer in a temporal FIFO file. These kind of pipes can be broken and result in a timeout or a freeze. The authors are supposedly familiar with the issue[153,154], and supply each release of GATK with the Conda YML file, listing the exact versions of the dependencies that work together. In a Conda environment or in a locally recreated environment, CNNScoreVariants works as expected.

DeepVariant requires a number of software packages installed, some of which might be unsuitable depending on the OS. The tool can be run via one of the official Docker images or the unofficial Conda recipe. Apart from the installation, the tool has been running as expected from the extensive practical documentation. It was possible to schedule and distribute the execution of a locally installed DeepVariant.

GARFIELD-NGS is the simplest tool of the observed, requiring only Perl and Java, both of which are present in most HPC systems. An additional shell script can be used to start several instances analyzing their part of the data.

VariationAnalysis is an immediate outlier from the other tools due to its infeasibility as a reliable variant calling utility. The tool did not produce any results apart from errors and, therefore, was discarded from the evaluation of efficiency and performance.

VQSR, again a part of GATK, but a relatively old one (introduced in 2013) and only relying on Java, processes the data in a well-tested, stable manner. The errors may originate from insufficient understanding or preparation of the data by the user: for example, it is recommended to perform filtering on a joint callset (from several samples), as it gives more power to the analysis. Given the age and the advanced technical support on the GATK forum, most issues are likely to be known or even solved.

Computational efficiency

The measurements were taken from runs of all presented tools against four WES samples (with original coverage), with respect to the required workflows from the alignment to the final variant callset file. Therefore, the runtime of filtering tools includes the runtime of HaplotypeCaller.

The boxplot below (Figure 8) shows the distribution of times spent on computation and from the perspective of the user, using the effective but not exhausting 8-core configuration. The ‘1 core’ notes indicate that the method or its part is not able or does not benefit from the use of more than 1 core. For instance, the last step of

DeepVariant is relatively simple (writing tensor records into text-based file) and only utilizes 1 thread (this is mentioned in the documentation and was verified); HaplotypeCaller in GATK4 is only able to utilize multiple cores if run in Spark or on Intel hardware with modern vector instructions (limited by 4).

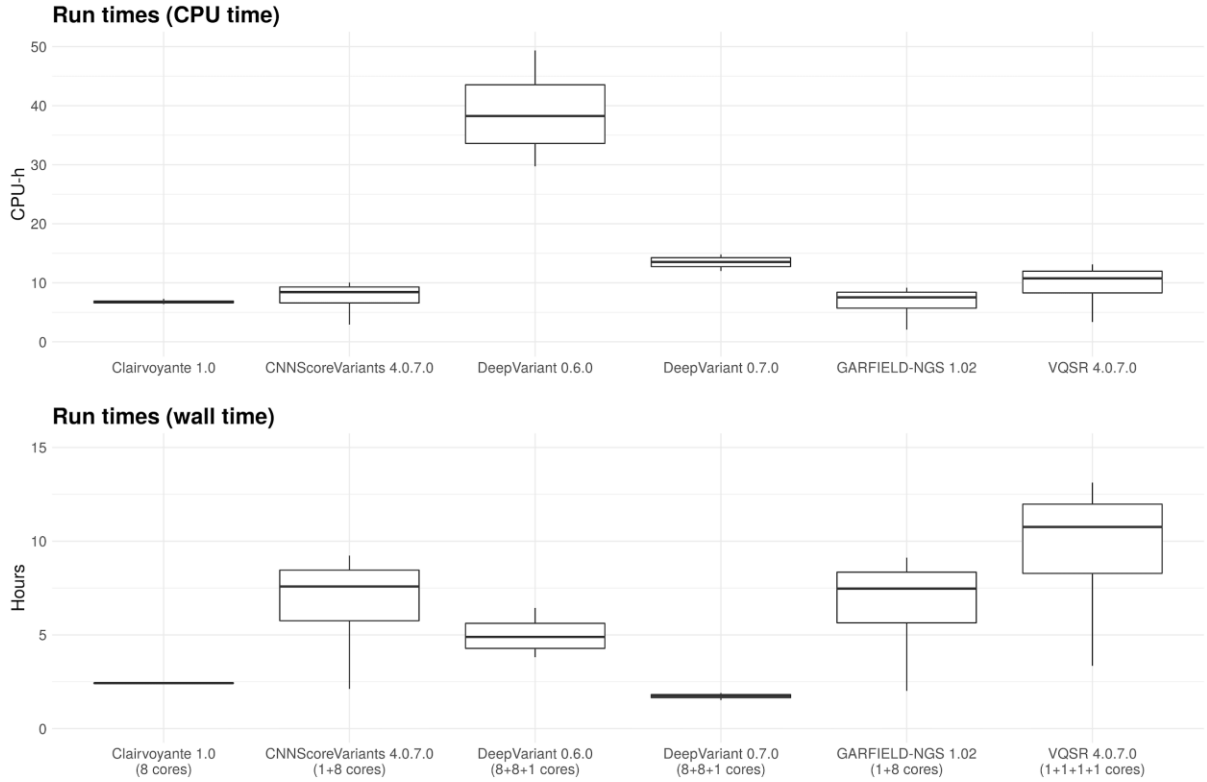


Figure 8. Run times comparison.

Clearly, the tools significantly differ in their use of computing resources. While the samples are supposed to be comparable, the tools may spend twice or even thrice more time on one sample over another (e.g. CNNScoreVariants, VQSR). For filtering tools, HaplotypeCaller noticeably contributes to the time and its variation, especially in the case of GARFIELD-NGS, that only takes several minutes on the cluster or several seconds on the laptop. Clairvoyante demonstrates little variation and good overall efficiency.

Two versions of DeepVariant were run, as the developers claimed that they have reduced the run time by 65% in v0.7.0. The claim is correct, although the performance in calling variants was slightly reduced as well.

Additionally, the CNN run times were measured for CNNScoreVariants and DeepVariant. The GPU instance was only utilized with them, because their models were the most demanding of the presented tools. On average, CNNScoreVariants is able to process one WES sample in 0.5 minutes, and one WGS sample in 13 minutes, while DeepVariant processes one WES sample in 2 minutes 30 seconds, and one WGS sample in 141 minutes.

For the memory usage, at their most, Clairvoyante have utilized 1.66 GB of RAM, CNNScoreVariants utilized 1.04 GB, one instance of parallelized DeepVariant 0.6.0 utilized 1.63 GB, one instance of DeepVariant 0.7.0 utilized 1.29 GB, VQSR was able to use 8.23 GB, and 'raw' HaplotypeCaller 9.17 GB.

For the disk storage usage, DeepVariant of both evaluated versions creates ~800 MB of intermediate files per sample, while the rest create temporary files of relatively smaller size (not exceeding the size of the VCF) in the specified or */tmp/user* directory.

Performance benchmark

The benchmarking procedures were performed according to the 'Evaluation' section of Chapter 5, with four Oslo University Hospital samples (WES) gradually downsampled to 10% of the original coverage. With these high-quality samples, the full range of downsampled samples is redundant, as the performance starts to significantly deteriorate only after the coverage reaches around ~50x (40% of the original coverage). Therefore the focus is on the lower coverages.

Figures 9-13 accommodate the PR-ROC curves (Chapter 5) for four samples in four rows, and for SNVs and indels in corresponding columns. Hence, it is possible to see how well the tools are able to work with the data of different origin.

In Figure 9, the differences between all tools but Clairvoyante are barely distinguishable with the naked eye. This is done on purpose: such scale demonstrates how poorly Clairvoyante performed. This is reasonable: the tool and the model were designed for the long read data, and while the selected model was trained on short read data (Illumina), the authors have used the hg38 reference for it, which is not completely compatible with the hs37d5 reference used in this study. The shape of the curves is interesting, too: Ashkenazi father and mother (HG003 and HG004) are more similar to each other than to their son (HG002), which has patterns that are close to that of Chinese son (HG005). This behaviour could have been a result of similarities in corresponding sequencing or alignment, yet the description of the data suggests equal parameters for all samples.

Clairvoyante is not included in further analysis. The remaining five out of six assessed methods performed well in SNV detection at high coverage, where precision and recall are known to be high with most modern approaches. In indel detection, the tools showed comparable performance at all coverages.

PR-ROC curves

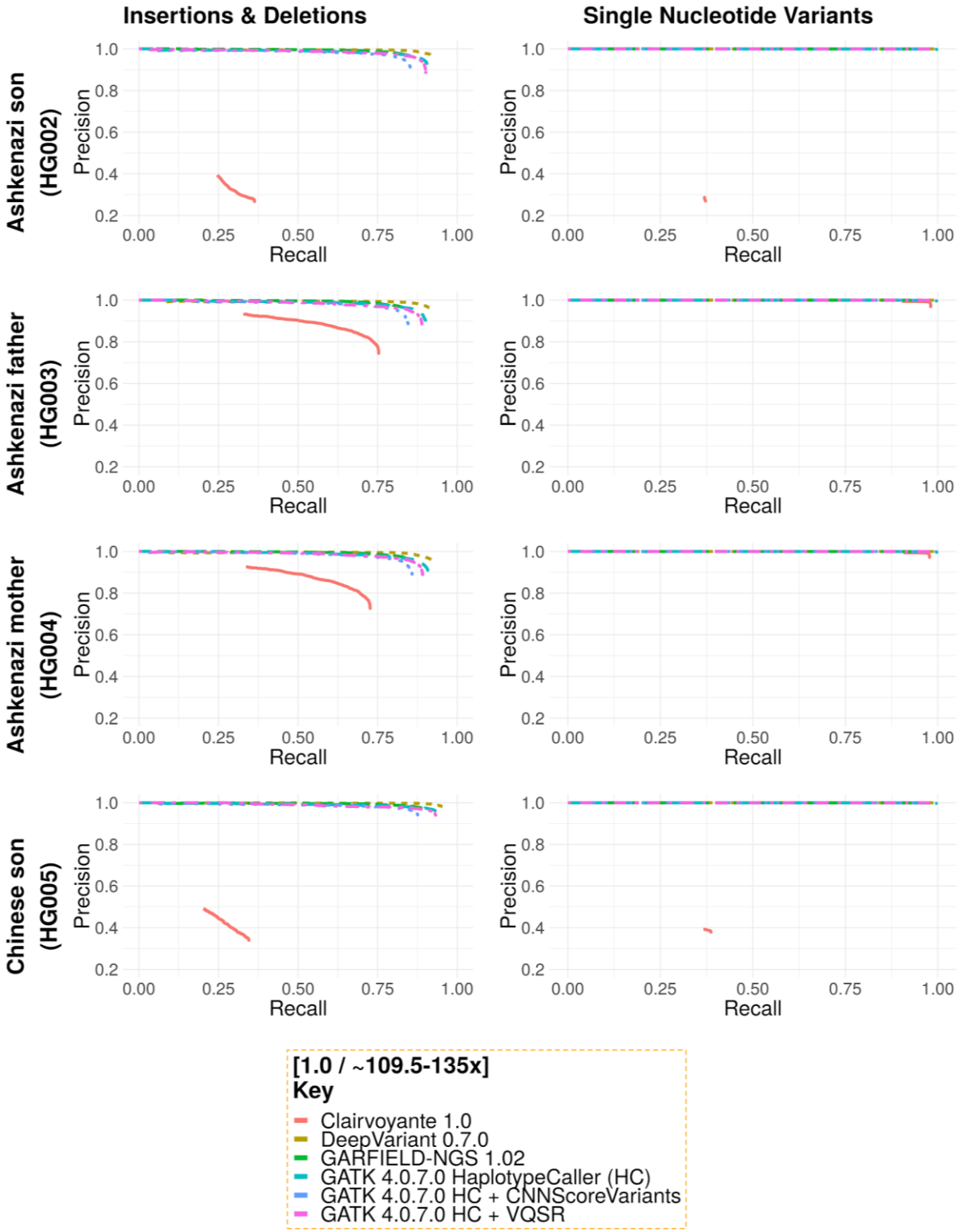


Figure 9. Precision and recall of calling SNVs and indels on four different samples. Small-scale overview showing all six methods at the original coverage.

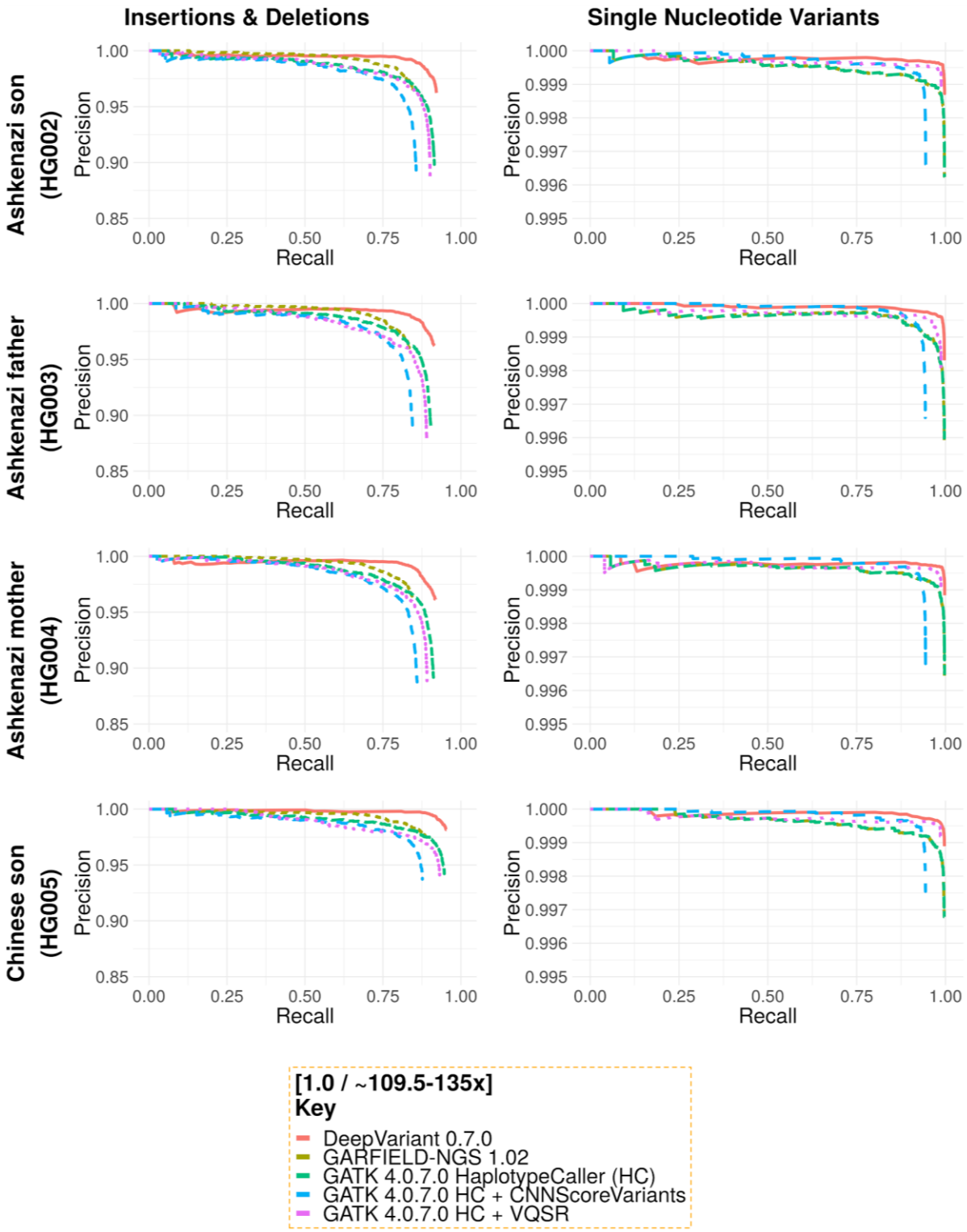


Figure 10. Precision and recall of calling SNVs and indels on four different samples. Close-up showing five methods at the original coverage (Clairvoyante is underperforming).

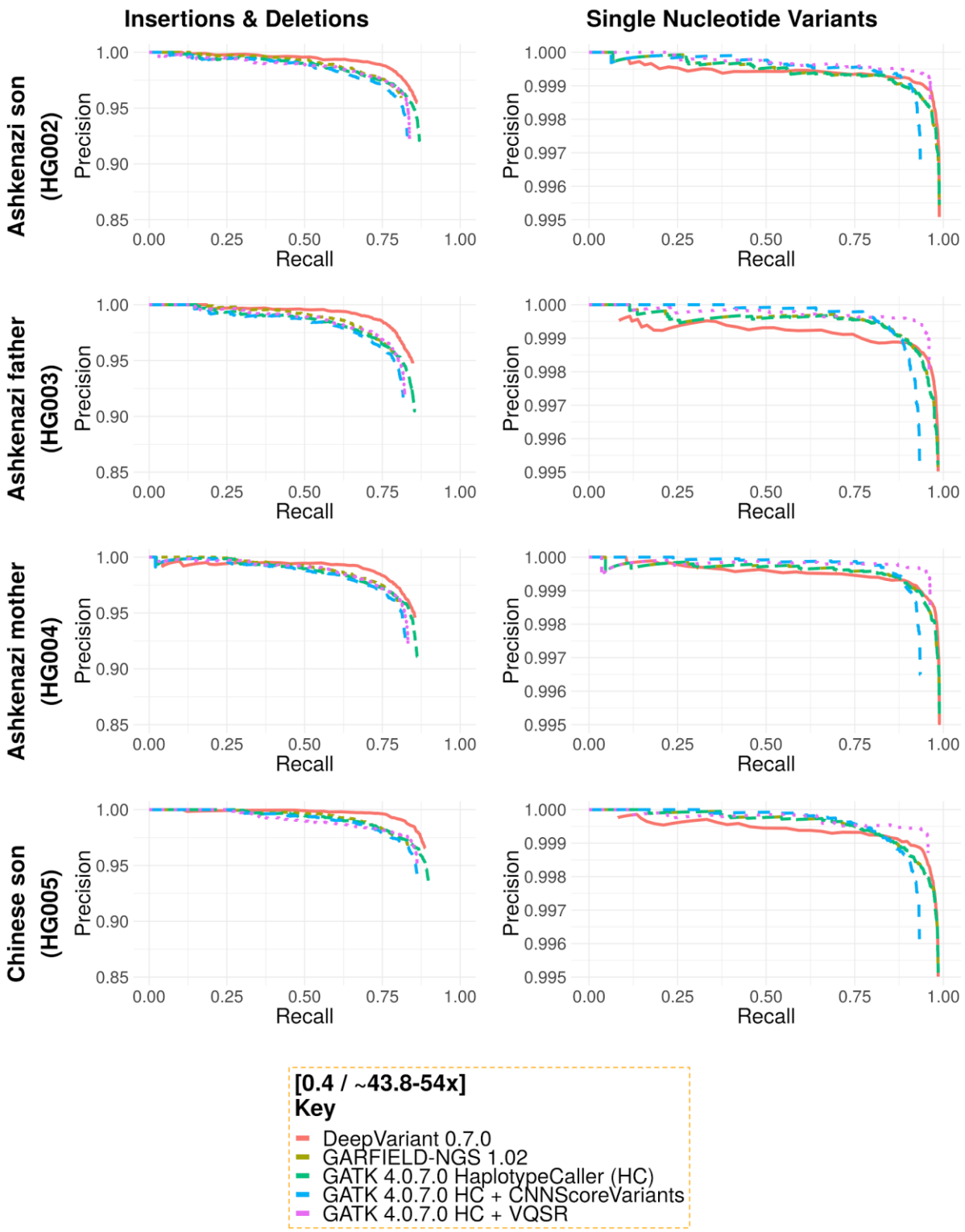


Figure 11. Precision and recall of calling SNVs and indels on four different samples. Close-up showing five methods at the 40% coverage. The precision starts to decrease at a notable pace.

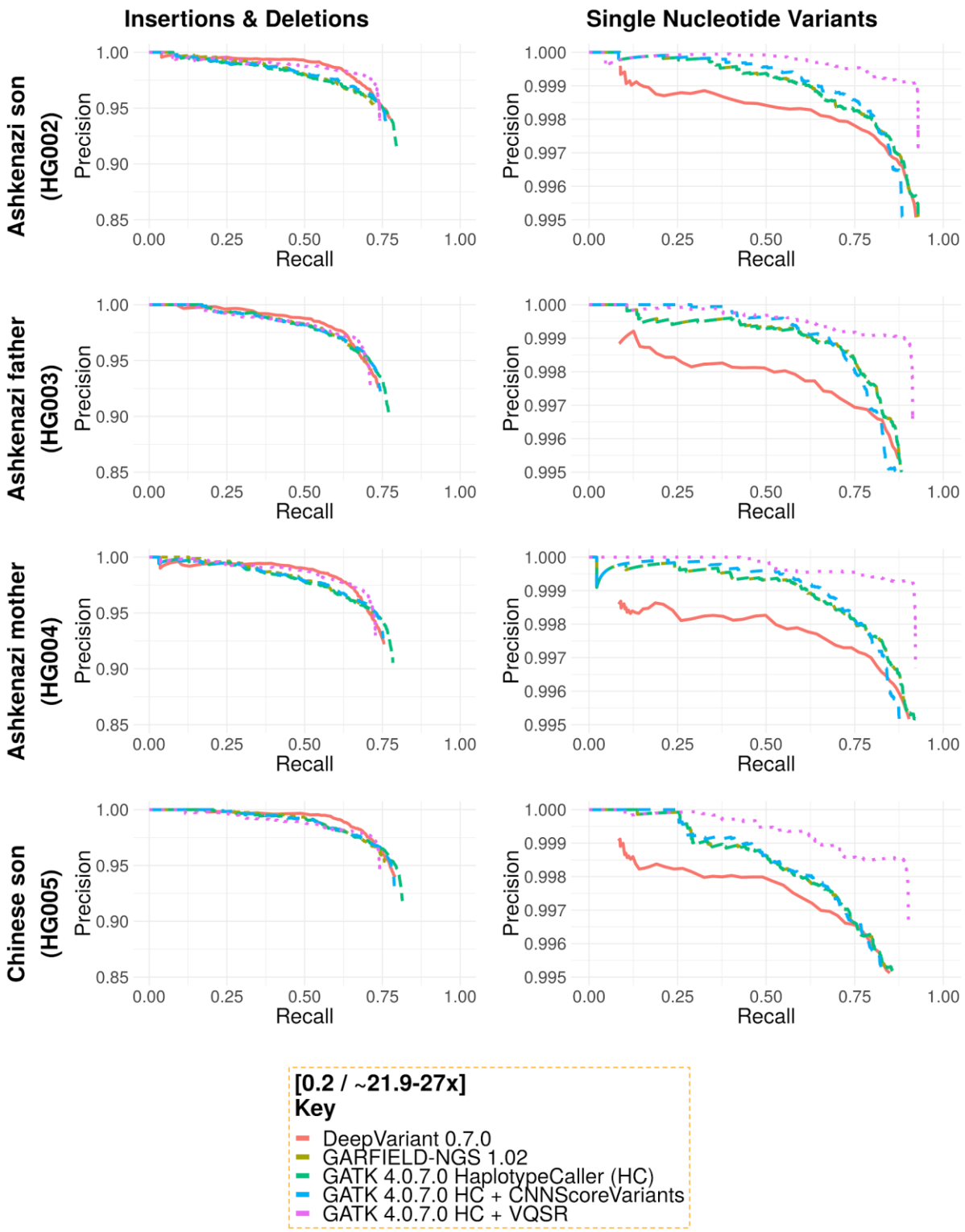


Figure 12. Precision and recall of calling SNVs and indels on four different samples. Close-up showing five methods at the 20% coverage. Indel calling seems to be more robust than SNV calling.

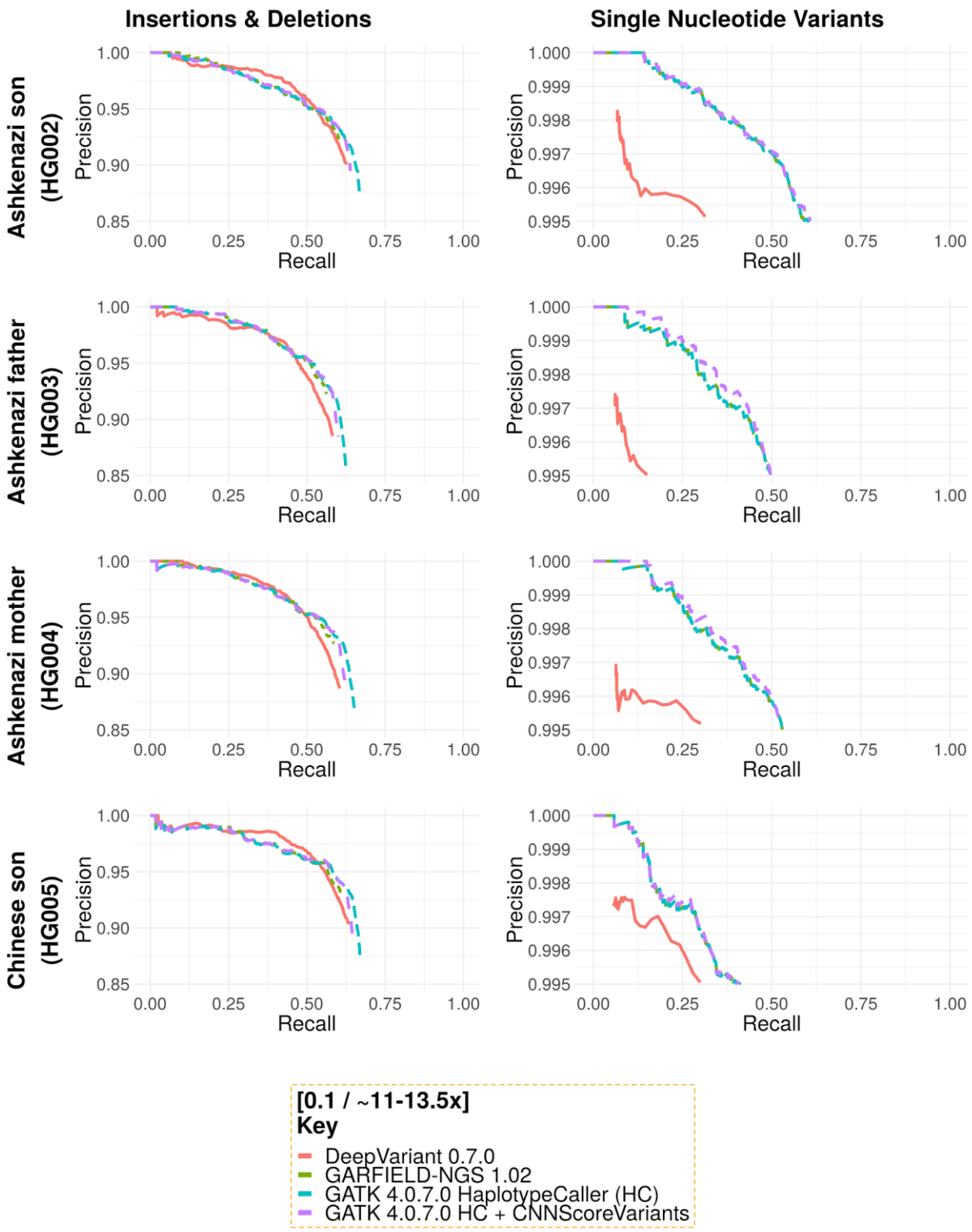


Figure 13. Precision and recall of calling SNVs and indels on four different samples. Close-up showing five methods at the 10% coverage. VQSR could not handle the filtering. DeepVariant in SNV detection suffered more than other tools.

At the original coverage, DeepVariant is the best performer on both indel and SNV calling. In indel detection, next is GARFIELD-NGS, showing improved (to 0.97 and higher) precision from 'raw' variants (HaplotypeCaller), at the expense of recall. In SNV detection, GARFIELD-NGS does not improve the result of HaplotypeCaller, practically repeating the curve points (0.998/0.99 and higher for the precision). CNNScoreVariants and VQSR improve on it, but worsen the precision/recall result of HaplotypeCaller for indel calling (0.95/0.825). CNNScoreVariants has lower recall than VQSR and pure HaplotypeCaller.

At 40% coverage, DeepVariant remains a clear winner in indel calling (0.97/0.77 and higher for the precision), but shows average to worst performance in SNV calling (0.998/0.95 and higher for the precision). Here, the benefit of applying the tools to the four samples instead of one (HG002 is commonly used in comparison and method studies (Refs)) is manifested: the curves are more consistent for Ashkenazi son (HG002) than for other samples. It is appropriate to think of *data leakage* - a machine learning scenario in which a model has access to the prediction data, usually due to a poor feature selection or cross-validation practice; however, without an unprotected access to the models and the data they were trained on, it is impossible to prove. Also at this point, VQSR shows slightly better indel calling precision over HaplotypeCaller (0.98/0.65 over 0.97/0.65 precision/recall) for Ashkenazi father and mother (HG003 and HG004), and better SNV calling precision and recall over all other tools for all samples (0.999/0.97 over 0.9985/0.96 closest). CNNScoreVariants shows better precision than HaplotypeCaller and DeepVariant for SNV calling (0.9995/0.8125), but otherwise for indel calling (0.96/0.75 against 0.97/0.75 closest). GARFIELD-NGS shows similar performance to that of VQSR in indel calling (0.97/0.75), and again follows the HaplotypeCaller curve in SNV calling.

At 20% coverage (the least sensible coverage for exome sequencing), DeepVariant shows better precision in indel calling than the rest, but concedes to VQSR in recall. The performances of CNNScoreVariants, GARFIELD-NGS and HaplotypeCaller in indel calling are very close, their curves are intermingled. In SNV calling, VQSR has superior precision and recall, GARFIELD-NGS and HaplotypeCaller form one curve again, relative to which CNNScoreVariants shows slightly better precision and slightly worse recall. DeepVariant does not perform so well in SNV calling. Still, for all tools, in SNV detection, recall is higher than 0.75 and precision is higher than 0.996; in indel detection, recall is higher than 0.7 and precision is higher than 0.95.

Figure 13 does not reflect a realistic scenario, but does give insight into variant calling in regions with small coverage, which can be found in a regular sequencing experiment, as well as into approaches to indel and SNV calling. Recall falls to 0.6 and lower in indel calling and to 0.5 and lower in SNV calling; precision remains higher than 0.85 in indel calling and higher than 0.995 in SNV calling. The curves of all tools but DeepVariant are close; DeepVariant shows recall lower than 0.25 in SNV calling. Filtering does not always work, VQSR could not process this low-coverage sampleset at all.

Apparently, indel detection is more tolerable to the quality of the data and more difficult to break; at the same time, the tools demonstrate better overall performance at higher coverages in SNV detection. Additionally, the tools are focused on improving precision rather than recall, which does not always make sense for a potential clinical setting: while it is definitely preferable to make correct calls, missing the other correct calls for this sake could disagree with the goal of screening for the known variants.[43,155]

Interestingly, the comparison of the callsets from tools with truths generated the following amounts of data points (variant conditions): from 889 to 890 for Clairvoyante, from 13729 to 15685 for CNNScoreVariants, from 297 to 301 for DeepVariant, from 13563 to 15478 for GARFIELD-NGS, from 29663 to 31130 for VQSR, and from 13729 to 15685 for the 'raw' HaplotypeCaller. This is congruous to the methods: the filtering solutions build on the output of HaplotypeCaller, adding or removing variant annotations, while solutions that work with alignments create their annotations from scratch, and less. The relatively small amount of data points from DeepVariant and Clairvoyante can be observed on the curves, given sufficiently large scale. A further study of the consistency of the final variants or errors could be justifiable; however, it is beyond the limits of this work.

Tables

The following tables summarize variant calling performance of the five tools (all neural network based tools but Clairvoyante, and VQSR) at the original coverage. The F1 metric and absolute numbers of TP, FP and FN variants are added.

The tables represent the whole callsets compared with truth, as opposed to plotting the metrics for variants that meet particular conditions.

Two DeepVariant versions were selected to show the difference; it is negligible. Three versions of CNNScoreVariants have been evaluated - 4.0.4.0, 4.0.5.0, 4.0.7.0; they show no difference in the results, hence only one (last) is selected.

Table 3. Summary evaluation of the performance of the tools on the HG002 sample.

Ashkenazi son (HG002)

	Tool	Precision	Recall	F1	TP	FP	FN
Insertions and Deletions	Clairvoyante 1.0	0.267223	0.36317	0.307895	1984	5446	3479
	DeepVariant 0.6.0	0.975299	0.924767	0.949361	5052	130	411
	DeepVariant 0.7.0	0.962307	0.921655	0.941542	5035	200	428
	GARFIELD-NGS 1.02	0.965553	0.845689	0.901655	4620	165	843
	GATK 4.0.7.0 HaplotypeCaller (HC)	0.898314	0.914699	0.906432	4997	573	466
	GATK 4.0.7.0 HC + CNNScoreVariants	0.89322	0.856306	0.874374	4678	567	785
	GATK 4.0.7.0 HC + VQSR	0.886408	0.901336	0.89381	4924	641	539
Single Nucleotide Variations	Clairvoyante 1.0	0.269517	0.372073	0.312599	18956	51388	31991
	DeepVariant 0.6.0	0.998644	0.997762	0.998203	50833	69	114
	DeepVariant 0.7.0	0.998703	0.997566	0.998134	50823	66	124
	GARFIELD-NGS 1.02	0.996234	0.997036	0.996635	50796	192	151
	GATK 4.0.7.0 HaplotypeCaller (HC)	0.996236	0.99735	0.996793	50812	192	135
	GATK 4.0.7.0 HC + CNNScoreVariants	0.996603	0.944295	0.969744	48109	164	2838
	GATK 4.0.7.0 HC + VQSR	0.998968	0.987811	0.993358	50326	52	621

Table 4. Summary evaluation of the performance of the tools on the HG003 sample.

Ashkenazi father (HG003)

	Tool	Precision	Recall	F1	TP	FP	FN
Insertions and Deletions	Clairvoyante 1.0	0.753256	0.753256	0.74898	4048	1386	1326
	DeepVariant 0.6.0	0.969738	0.914589	0.941356	4915	156	459
	DeepVariant 0.7.0	0.962155	0.915147	0.938062	4918	196	456
	GARFIELD-NGS 1.02	0.960368	0.834016	0.892743	4482	185	892
	GATK 4.0.7.0 HaplotypeCaller (HC)	0.890879	0.90361	0.8972	4856	603	518
	GATK 4.0.7.0 HC + CNNScoreVariants	0.885523	0.846855	0.865757	4856	597	823
	GATK 4.0.7.0 HC + VQSR	0.878454	0.89077	0.884569	4787	673	587
Single Nucleotide Variations	Clairvoyante 1.0	0.969153	0.980858	0.97497	49344	1571	963
	DeepVariant 0.6.0	0.998686	0.997138	0.997911	50163	66	144
	DeepVariant 0.7.0	0.998309	0.996919	0.997613	50152	85	155
	GARFIELD-NGS 1.02	0.995927	0.996422	0.996175	50127	205	180
	GATK 4.0.7.0 HaplotypeCaller (HC)	0.99591	0.99682	0.996364	50147	206	160
	GATK 4.0.7.0 HC + CNNScoreVariants	0.996598	0.943169	0.969148	47448	162	2859
	GATK 4.0.7.0 HC + VQSR	0.998051	0.987318	0.992656	49669	97	638

Table 5. Summary evaluation of the performance of the tools on the HG004 sample.

Ashkenazi mother (HG004)

	Tool	Precision	Recall	F1	TP	FP	FN
Insertions and Deletions	Clairvoyante 1.0	0.726943	0.726514	0.726728	4006	1504	1508
	DeepVariant 0.6.0	0.971186	0.921473	0.945677	5081	153	433
	DeepVariant 0.7.0	0.961596	0.918571	0.939591	5065	205	449
	GARFIELD-NGS 1.02	0.964167	0.844033	0.90011	4654	173	860
	GATK 4.0.7.0 HaplotypeCaller (HC)	0.891693	0.911861	0.901664	5028	618	486
	GATK 4.0.7.0 HC + CNNScoreVariants	0.859449	0.887553	0.873275	4739	608	775
	GATK 4.0.7.0 HC + VQSR	0.888671	0.891186	0.889927	4914	624	600
Single Nucleotide Variations	Clairvoyante 1.0	0.971673	0.977673	0.974664	49656	1448	1134
	DeepVariant 0.6.0	0.998758	0.997559	0.998158	50666	63	124
	DeepVariant 0.7.0	0.998837	0.997657	0.998247	50671	59	119
	GARFIELD-NGS 1.02	0.99644	0.997224	0.996832	50649	181	141
	GATK 4.0.7.0 HaplotypeCaller (HC)	0.996441	0.9975	0.99697	50663	181	127
	GATK 4.0.7.0 HC + CNNScoreVariants	0.996737	0.943926	0.969613	47942	157	2848
	GATK 4.0.7.0 HC + VQSR	0.999024	0.987419	0.993188	50151	49	639

Table 6. Summary evaluation of the performance of the tools on the HG005 sample.

Chinese son (HG005)

	Tool	Precision	Recall	F1	TP	FP	FN
Insertions and Deletions	Clairvoyante 1.0	0.33942	0.345083	0.342228	1674	3256	3177
	DeepVariant 0.6.0	0.983151	0.950526	0.966563	4611	80	240
	DeepVariant 0.7.0	0.981528	0.953206	0.96716	4624	88	227
	GARFIELD-NGS 1.02	0.975306	0.88724	0.929191	4304	109	547
	GATK 4.0.7.0 HaplotypeCaller (HC)	0.940081	0.947021	0.943538	4594	296	257
	GATK 4.0.7.0 HC + CNNScoreVariants	0.937255	0.87652	0.905871	4252	288	599
	GATK 4.0.7.0 HC + VQSR	0.940246	0.932797	0.936507	4525	291	326
Single Nucleotide Variations	Clairvoyante 1.0	0.378428	0.386692	0.382515	19968	32801	31670
	DeepVariant 0.6.0	0.998836	0.99694	0.997887	51480	60	158
	DeepVariant 0.7.0	0.998894	0.997076	0.997984	51487	57	151
	GARFIELD-NGS 1.02	0.996784	0.996204	0.996494	51442	166	196
	GATK 4.0.7.0 HaplotypeCaller (HC)	0.996785	0.996417	0.996601	51453	166	185
	GATK 4.0.7.0 HC + CNNScoreVariants	0.997361	0.943995	0.969944	48746	129	2892
	GATK 4.0.7.0 HC + VQSR	0.999078	0.985708	0.992348	50900	47	738

In indel calling, the tools are not always able to surpass 0.9 mark of the F1 score. At the same time, in SNV calling the tools are most often improving 0.99 mark of F1.

It is clear that Clairvoyante is out of place in this comparison; yet for two samples (HG003 and HG004) it can outperform CNNScoreVariants. The comparison using an alignment to a more suitable reference (hg38) could yield more practical results; however, it would require a new analysis of other tools and as such, is beyond the limits of this work.

CNNScoreVariants, in turn, would benefit from a more sensitive filtering. In addition, the analysis of a high-quality sample such as the ones that were used, does seem to marginally reduce the amount of FP variants, but takes down the amount of TP variants as well, and increases the amount of FN variants.

DeepVariant demonstrates outstanding performance in indel calling, where the variation of outcome is high. The newer version of its model may or may not improve on the old; the significantly reduced computational load makes it a reasonable update.

GARFIELD-NGS does filter the output of HaplotypeCaller in indel calling, gaining precision at the expense of recall. In SNV calling, its model does not significantly improve, but does not significantly degrade the result either.

Finally, for SNVs, VQSR retains a level of filtering lower that other calls, but still with 0.99+ range. For indels, it filters out too much in both precision and recall, worsening the result of HaplotypeCaller. VQSR works noticeably better on lower coverages.

Chapter 7: Conclusions

The aim of this work was to test whether the neural network based methods for variant calling are applicable in the given pipeline conditions. In doing so, the theoretical background had to be leveraged and the workflow of the project had to be designed and followed accordingly; the practical solutions had to be selected, built, and applied.

At first, the knowledge regarding the state of variant detection had been established. It spans from the basics of genetics and sequencing technologies to the principles of handling genomic data to the machine learning concepts and techniques. The inspection of the deep learning tools had shed the light on the architectural details and development history of their models; it did not, however, aid in interpretation of models as it is a far bigger issue. The latter could have been easier given the means for the interaction with or visualization of the model (which is highly unrealistic) or given the elaborate and consistent documentation from the authors of the particular methods (which is less unrealistic).

Secondly, the search for and preparation of the appropriate data revealed the the lack of the publicly available sequencing data, reflecting the real-world rather than ideal application, coupled with the variant callset representing the ground truth. As of this writing, only three such truth sets exist, giving room for the potential bias, but also for the improvement. The exome data corresponding to them is not abundant, even though at least one sample (HG001 / RM 8398) is intended to be routinely included as control in the genomic studies. This particular sample, however, was often used for the training, and therefore is not the best possible choice for the testing.

Thirdly, the installation and the usage of the currently available deep learning based tools exposed the heterogeneity of the approaches to their development and utilization. Certain callers, such as DeepVariant, are clearly intended to be run on the powerful, distributed production systems, while others, such as GARFIELD-NGS or GATK, are suitable for the desktop environment. A practical workflow for each of the working tools have been extracted or devised and the required steps, commands, and options have been documented. The convenience, the turnaround time and the computational efficiency criteria were taken into account during the evaluation.

Lastly, the evaluation of the performance showed several interesting and thought-provoking results. For instance, why would VQSR and CNNScoreVariants decrease both the precision and the recall at the high coverage? One reason for VQSR could be a poor statistical power from joint genotyping of only 4 samples; yet at lower coverages VQSR performs well or even better than the others. One reason for the aforementioned behaviour in CNNScoreVariants could be a weak generalization to different conditions with the pre-trained model. These conclusions are premature.

Another interesting observation is that at high coverages, DeepVariant showed significant improvement in precision and a generally high score in recall metric at the expense of using more computational resources (which correlates with its immensely deep model). And yet another observation is that GARFIELD-NGS performs fairly well on indels, despite the initial apprehension from the unusually fast execution, but does not improve SNV calling by large. All observations are listed in Chapter 6.

The comparisons, evaluations and reviews of variant callers had been conducted before. The advantage of this work over similar studies are in the theoretical and practical detail, as well as in the range of samples used for the evaluation.

The desired result here was the eventual augmentation of the established variant calling pipeline. Several tools proved to be reliable in given conditions; certain others (Clairvoyante, VariationAnalysis) did not, but nonetheless gave points to the potential further development of the pipeline, the methods that it utilizes, and the effect that it may have on understanding of the genetic variation.

References

1. Homo sapiens (ID 51) - Genome - NCBI [Internet]. Available from: https://www.ncbi.nlm.nih.gov/genome/51
2. Elgar G, Vavouri T. Tuning in to the signals: noncoding sequence conservation in vertebrate genomes. Trends Genet. 2008;24(7):344–52.
3. Alberts B, et al. Molecular Biology of the Cell. 6th ed. W. W. Norton & Company; 2014.
4. Sherry ST. dbSNP: the NCBI database of genetic variation. Nucleic Acids Res. 2001;29(1):308–11.
5. rs334 RefSNP Report - dbSNP - NCBI [Internet]. Available from: https://www.ncbi.nlm.nih.gov/snp/rs334
6. Krusche P, Trigg L, Boutros PC, Mason CE, De FM, Vega L, et al. Best Practices for Benchmarking Germline Small Variant Calls in Human Genomes. 2018; Available from: http://dx.doi.org/10.1101/270157
7. Hu J, Ng PC. Predicting the effects of frameshifting indels. Genome Biol. 2012;13(2).
8. De La Chaux N, Messer PW, Arndt PF. DNA indels in coding regions reveal selective constraints on protein evolution in the human lineage. BMC Evol Biol. 2007;7.
9. Gemayel R, Vinces MD, Legendre M, Verstrepen KJ. Variable Tandem Repeats Accelerate Evolution of Coding and Regulatory Sequences. Annu Rev Genet. 2010;44(1):445–77.
10. Richard G-F, Kerrest A, Dujon B. Comparative Genomics and Molecular Dynamics of DNA Repeats in Eukaryotes. Microbiol Mol Biol Rev. 2008;72(4):686–727.
11. Sharp AJ, Zeng H, Guilmatre A, Georgiev S, Price AL, Erlich Y, et al. Abundant contribution of short tandem repeats to gene expression variation in humans. Nat Genet. 2015;48(1):22–9.
12. Myers RH. Huntington’s Disease Genetics. NeuroRx. 2004;1(2):255–62.
13. Langbehn DR, Stout JC, Scahill RI, Long JD, Wexler A, Warner JH, et al. Huntington disease: natural history, biomarkers and prospects for therapeutics. Nat Rev Neurol. 2014;10(4):204–16.
14. Moss DJH, Tabrizi SJ, Mead S, Lo K, Pardiñas AF, Holmans P, et al. Identification of genetic variants associated with Huntington’s disease

progression: a genome-wide association study. Lancet Neurol. 2017;16(9):701–11.

15. Bates GP, Dorsey R, Gusella JF, Hayden MR, Kay C, Leavitt BR, et al. Huntington disease. Nat Rev Dis Prim [Internet]. Macmillan Publishers Limited; 2015 Apr 23;1:15005. Available from: https://doi.org/10.1038/nrdp.2015.5

16. Theilmann J, Squitieri F, Kremer B, Zeisler J, Telenius H, Bird TD, et al. A Worldwide Study of the Huntington's Disease Mutation: The Sensitivity and Specificity of Measuring CAG Repeats. N Engl J Med. 2002;330(20):1401–6.

17. Feuk L, Carson AR, Scherer SW. Structural variation in the human genome. Nat Rev Genet. 2006;7(2):85–97.

18. Alkan C, Coe BP, Eichler EE. Genome structural variation discovery and genotyping. Nat Rev Genet. 2011;12(5):363–76.

19. Spielmann M, Lupiáñez DG, Mundlos S. Structural variation in the 3D genome. Nat Rev Genet. 2018;19(7):453–67.

20. Tucker T, Marra M, Friedman JM. Massively Parallel Sequencing: The Next Big Thing in Genetic Medicine. Am J Hum Genet. 2009;85(2):142–54.

21. Ramos L. Miodrag Mićić (Ed): Sample Preparation Techniques for Soil, Plant and Animal Samples. Chromatographia. 2016;79(23–24):1683–4.

22. Mullegama S V., Alberti MO, Au C, Li Y, Toy T, Tomasian V, et al. Nucleic acid extraction from human biological samples. Methods Mol Biol. 2019;1897:359–83.

23. Ali N, Rampazzo R de CP, Costa ADT, Krieger MA. Current Nucleic Acid Extraction Methods and Their Implications to Point-of-Care Diagnostics. Biomed Res Int. 2017;2017:1–13.

24. Mamanova, L., Coffey, A. J., Scott, C. E., Kozarewa, I., Turner, E. H., Kumar, A., … Turner, D. J. (2010). Target-enrichment strategies for next-generation sequencing. Nature Methods, 7(2), 111–118. https://doi.org/10.1038/nmeth.1419

25. Teer JK, Mullikin JC. Exome sequencing: The sweet spot before whole genomes. Hum Mol Genet. 2010;19(R2).

26. Watson M, Hume D, Archibald A, Deeb N, Warr A, Robert C. Exome Sequencing: Current and Future Perspectives. G3: Genes|Genomes|Genetics. 2015;5(8):1543–50.

27. Horn G, Saiki R, Erlich H, Faloona F, Scharf S, Mullis K. Specific Enzymatic Amplification of DNA In Vitro: The Polymerase Chain Reaction. Cold Spring Harb Symp Quant Biol. 1986;51(0):263–73.

28. Pray LA. The Biotechnology Revolution: PCR and the Use of Reverse Transcriptase to Clone Expressed Genes [Internet]. Scitable. 2008. Available from: https://www.nature.com/scitable/topicpage/the-biotechnology-revolution-pcr-and-the-use-553

29. Metzker ML. Sequencing technologies — the next generation. Nat Rev Genet. 2010;11(1):31–46.

30. Heather JM, Chain B. The sequence of sequencers: The history of sequencing DNA. Genomics. 2016;107(1):1–8.

31. Mardis ER. DNA sequencing technologies: 2006-2016. Nat Protoc. 2017;12(2):213–8.

32. Novo SM, Banerjee S, Benoit VA, Rasolonjatovo IMJ, Bridgham JA, Golda GS, et al. Accurate whole human genome sequencing using reversible terminator chemistry. Nature. 2008;456(7218):53–9.

33. Ronaghi M, Uhlén M, Nyrén P. A sequencing method based on real-time pyrophosphate. Science (80-). 1998;281(5375):363–5.

34. Volkmer GA, Irzyk GP, Gomes X V., Makhijani VB, Roth GT, Plant R, et al. Genome sequencing in microfabricated high-density picolitre reactors. Nature. 2005;437(7057):376–80.

35. Landegren U, Kaiser R, Sanders J, Hood L. A ligase-mediated gene detection technique. Science (80-). 1988;241(4869):1077–80.

36. Sidow A, Stuart J, McKernan K, Ichikawa J, Tonthat T, Zeng K, et al. A high-resolution, nucleosome position map of C. elegans reveals a lack of universal sequence-dictated positioning. Genome Res. 2008;18(7):1051–63.

37. Schadt EE, Turner S, Kasarskis A. A window into third-generation sequencing. Hum Mol Genet. 2010;19(R2).

38. Turner S, Otto G, Murphy D, Luong K, Eid J, Lundquist P, et al. Real-Time DNA Sequencing from Single Polymerase Molecules. Science (80-). 2008;323(5910):133–8.

39. Clarke J, Wu HC, Jayasinghe L, Patel A, Reid S, Bayley H. Continuous base identification for single-molecule nanopore DNA sequencing. Nat Nanotechnol. 2009;4(4):265–70.

40. McKenna A, Hanna M, Banks E, Sivachenko A, Cibulskis K, Kernytsky A, et al. The genome analysis toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. Genome Res. 2010;20(9):1297–303.

41. DePristo MA, Rivas MA, McKenna A, Hartl C, del Angel G, Sivachenko AY, et al. A framework for variation discovery and genotyping using next-generation DNA sequencing data. Nat Genet. 2011;43(5):491–8.

42. Ross MG, Russ C, Costello M, Hollinger A, Lennon NJ, Hegarty R, et al. Characterizing and measuring bias in sequence data. Genome Biol. 2013;14(5).

43. Roy S, Coldren C, Karunamurthy A, Kip NS, Klee EW, Lincoln SE, et al. Standards and Guidelines for Validating Next-Generation Sequencing Bioinformatics Pipelines: A Joint Recommendation of the Association for Molecular Pathology and the College of American Pathologists. J Mol Diagnostics. 2018;20(1):4–27.

44. Wang K, Li M, Hakonarson H. ANNOVAR: Functional annotation of genetic variants from high-throughput sequencing data. Nucleic Acids Res. 2010;38(16).

45. Butkiewicz M, Blue EE, Leung YY, Jian X, Marcora E, Renton AE, et al. Functional annotation of genomic variants in studies of late-onset Alzheimer's disease. Bioinformatics. 2018;34(16):2724–31.

46. Malheiro A, Riley G, Hart J, Jang W, Ovetsky M, Hoffman D, et al. ClinVar: improving access to variant interpretations and supporting evidence. Nucleic Acids Res. 2017;46(D1):D1062–7.

47. Kulkarni S, Roy S, Younes A, Lindeman NI, Duncavage EJ, Datto M, et al. Standards and Guidelines for the Interpretation and Reporting of Sequence Variants in Cancer. J Mol Diagnostics. 2016;19(1):4–23.

48. Shirts BH, Pritchard CC, Walsh T. Family-Specific Variants and the Limits of Human Genetics. Trends Mol Med. 2016;22(11):925–34.

49. Lubitz SA, Haas ME, Khera A V., Aragam KG, Kathiresan S, Ellinor PT, et al. Genome-wide polygenic scores for common diseases identify individuals with risk equivalent to monogenic mutations. Nat Genet. 2018;50(9):1219–24.

50. Jain P, O'Roak BJ, Cooper GM, Witten DM, Shendure J, Kircher M. A general framework for estimating the relative pathogenicity of human genetic variants. Nat Genet. 2014;46(3):310–5.

51. Smith LD, Herd S, Willig LK, Marrs T, Walter A, Thiffault I, et al. A 26-hour system of highly sensitive whole genome sequencing for emergency management of genetic diseases. Genome Med. 2015;7(1).

52. Kalman L, Liu CSJ, Zook JM, Reese MG, Johnson PLF, Voelkerding K V, et al. Assuring the quality of next-generation sequencing in clinical laboratory practice. Nat Biotechnol. 2012;30(11):1033–6.

53. Matthijs G, Swinnen E, Müller CR, Corveleyn A, Wallace A, Mattocks CJ, et al. A standardized framework for the validation and verification of clinical molecular genetic tests. Eur J Hum Genet. 2010;18(12):1276–88.

54. Truitt Cho M, Wynn J, Iglesias A, Anyane-Yeboa K, Chung WK, Guzman E, et al. The usefulness of whole-exome sequencing in routine clinical practice. Genet Med. 2014;16(12):922–31.

55. Poplin R, Ruano-Rubio V, DePristo MA, Fennell TJ, Carneiro MO, Auwera GA Van der, et al. Scaling accurate genetic variant discovery to tens of thousands of samples. bioRxiv [Internet]. 2017;201178. Available from: https://www.biorxiv.org/content/early/2017/11/14/201178.1

56. Smith TF, Waterman MS. Identification of common molecular subsequences. J Mol Biol. 1981;147(1):195–7.

57. Durbin R, Eddy S, Krogh A, Mitchison G. Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids. Combridge University Press. 1998.

58. Eddy SR. What is a hidden Markov model? Nat Biotechnol. 2004;22(10):1315–6.

59. Garimella K V., Levy-Moonshine A, Jordan T, Van der Auwera GA, Hartl C, del Angel G, et al. From FastQ Data to High-Confidence Variant Calls: The Genome Analysis Toolkit Best Practices Pipeline. Curr Protoc Bioinforma. 2013;11.10.1-11.10.33.

60. Poplin R. Variant Quality Score Recalibration (VQSR) — GATK-Forum [Internet]. 2012. Available from: https://gatkforums.broadinstitute.org/gatk/discussion/39/variant-quality-score-recalibration-vqsr

61. GATK | Doc #11084 | Variant Quality Score Recalibration (VQSR) [Internet]. 2017. Available from: https://software.broadinstitute.org/gatk/documentation/article?id=11084

62. Zook J, McDaniel J, Parikh H, Heaton H, Irvine SA, Trigg L, et al. Reproducible integration of multiple sequencing datasets to form high-confidence SNP, indel, and reference calls for five human genome reference materials. bioRxiv [Internet]. 2018;281006. Available from: https://www.biorxiv.org/content/early/2018/05/25/281006.full.pdf+html

63. Reed R, Marks II RJ. Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks. The MIT Press; 1999.

64. Marsland S. Machine Learning: An Algorithmic Perspective. 2nd ed. Chapman and Hall/CRC; 2014.

65. Veen F. The Neural Network Zoo - The Asimov Institute [Internet]. 2016. Available from: http://www.asimovinstitute.org/neural-network-zoo/

66. Goodfellow I, Bengio Y, Courville A. Deep Learning [Internet]. MIT Press; 2016. Available from: https://www.deeplearningbook.org/

67. Lipton ZC, Berkowitz J, Elkan C. A Critical Review of Recurrent Neural Networks for Sequence Learning. 2015; Available from: http://arxiv.org/abs/1506.00019

68. Hochreiter S, Schmidhuber J. Long Short-Term Memory. Neural Comput. 1997;9(8):1735–80.

69. Olah C. Understanding LSTM Networks -- colah's blog [Internet]. 2015. Available from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

70. Ioffe S, Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015; Available from: http://arxiv.org/abs/1502.03167

71. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. J Mach Learn Res [Internet]. 2014;15:1929–58. Available from: http://jmlr.org/papers/v15/srivastava14a.html

72. Jansma H. Don't Use Dropout in Convolutional Networks. – Towards Data Science [Internet]. 2018. Available from: https://towardsdatascience.com/dont-use-dropout-in-convolutional-networks-81486c823c16

73. Luo R, Sedlazeck FJ, Lam T-W, Schatz MC. A multi-task convolutional deep neural network for variant calling in single molecule sequencing. Nat Commun [Internet]. 2019;10(1):998. Available from: https://doi.org/10.1038/s41467-019-09025-z

74. Friedman S. Deep learning in GATK4 [Internet]. GATK | Blog. 2017. Available from: https://software.broadinstitute.org/gatk/blog?id=10996

75. Ravasio V, Ritelli M, Legati A, Giacopuzzi E. GARFIELD-NGS: Genomic vARiants FIltering by dEep Learning moDels in NGS. Bioinformatics. 2018;34(17):3038–40.

76. Dijamco J, DePristo MA, McLean CY, Afshar PT, Colthurst T, Newburger D, et al. A universal SNP and small-indel variant caller using deep neural networks. Nat Biotechnol. 2018;

77. Torracinta RR, Campagne F. Training Genotype Callers with Neural Networks. bioRxiv [Internet]. 2016;097469+. Available from: http://dx.doi.org/10.1101/097469

78. Klambauer G, Unterthiner T, Mayr A, Hochreiter S. Self-Normalizing Neural Networks. 2017; Available from: http://arxiv.org/abs/1706.02515

79. Zook JM, Catoe D, McDaniel J, Vang L, Spies N, Sidow A, et al. Extensive sequencing of seven human genomes to characterize benchmark reference materials. Sci Data. 2016;3.

80. Zook JM, Chapman B, Wang J, Mittelman D, Hofmann O, Hide W, et al. Integrating human sequence data sets provides a resource of benchmark SNP and indel genotype calls. Nat Biotechnol. 2014;32(3):246–51.

81. Friedman S. CNN deep learning pipeline out of beta! [Internet]. GATK | Blog. 2019. Available from: https://software.broadinstitute.org/gatk/blog?id=23457

82. Friedman S. MIA: Primer: Classifying genomic sequences with convolutional neural networks [Internet]. 2017. Available from: https://www.youtube.com/watch?v=jWDnK-CLIzk

83. Friedman S. Broad Genomics Community Meeting: Deep Learning with Convolutional Neural Networks for Variant Filtering and Calling. 2018; Available from: https://www.youtube.com/watch?v=y5to43kNmkY

84. Friedman S. MIA: Variant Filtering and Calling with Convolutional Neural Networks. 2018; Available from: https://www.youtube.com/watch?v=vWmepxBi0kI

85. Friedman S. MIA: Deep Learning Convolutions for Variant Filtration. 2019; Available from: https://www.youtube.com/watch?v=gWcFJiYZNZ0

86. CNNScoreVariants [Internet]. GATK | Tool Documentation. Available from: https://software.broadinstitute.org/gatk/documentation/tooldocs/4.0.7.0/org_broadinstitute_hellbender_tools_walkers_vqsr_CNNScoreVariants.php

87. GitHub - broadinstitute/gatk: Official code repository for GATK versions 4 and up [Internet]. Available from: https://github.com/broadinstitute/gatk/tree/4.0.7.0

88. Farjoun Y. What is truth? Or, how an accident of nature can illuminate our path [Internet]. GATK | Blog. 2017. Available from: https://software.broadinstitute.org/gatk/blog?id=10912

89. Fleharty M, Bloom JM, MacArthur D, Li H, Neale B, Gauthier L, et al. A synthetic-diploid benchmark for accurate variant-calling evaluation. Nat Methods. 2018;15(8):595–7.

90. Eberle MA, Fritzilas E, Krusche P, Källberg M, Moore BL, Bekritsky MA, et al. A reference data set of 5.4 million phased human variants validated by genetic inheritance from sequencing a three-generation 17-member pedigree. Genome Res. 2017;27(1):157–64.

91. Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the Inception Architecture for Computer Vision. 2015; Available from: http://arxiv.org/abs/1512.00567

92. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, et al. Going deeper with convolutions. Proc IEEE Comput Soc Conf Comput Vis Pattern Recognit. 2015;07–12–June:1–9.

93. PrecisionFDA Truth Challenge [Internet]. 2016. Available from: https://precision.fda.gov/challenges/truth

94. Variant Calling on a Rice genome with DeepVariant [Internet]. Google Codelabs. Available from: https://codelabs.developers.google.com/codelabs/genomics-deepvariant/#0

95. The H2O.ai team. H2O: Scalable Machine Learning [Internet]. 2015. Available from: http://www.h2o.ai

96. GATK | Doc #1259 | Which training sets / arguments should I use for running VQSR? [Internet]. 2012. Available from: https://software.broadinstitute.org/gatk/documentation/article.php?id=1259

97. Anderson C. Docker [Software engineering]. IEEE Softw. 2015;32(3):102-c3.

98. Conda documentation [Internet]. 2018. Available from: https://docs.conda.io/en/latest/

99. Whitepaper: NVIDIA Tesla P100: The Most Advanced Datacenter Accelerator Ever Built [Internet]. 2016. Available from: https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

100. Nickolls J, Buck I, Garland M, Skadron K. Scalable parallel programming with CUDA. Queue. 2008;6(2):40.

101. cuDNN Installation Guide [Internet]. [cited 2018 Jul 17]. Available from: https://docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html

102. NVIDIA Docker: GPU Server Application Deployment Made Easy [Internet]. NVIDIA Developer Blog. 2016. Available from: https://devblogs.nvidia.com/nvidia-docker-gpu-server-application-deployment-made-easy/

103. Lipman DJ, Pearson WR. Rapid and sensitive protein similarity searches. Science (80-). 1985;227(4693):1435–41.

104. Pearson WR, Lipman DJ. Improved tools for biological sequence comparison. Proc Natl Acad Sci. 2006;85(8):2444–8.

105. Cock PJA, Fields CJ, Goto N, Heuer ML, Rice PM. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. Nucleic Acids Res. 2009;38(6):1767–71.

106. The SAM/BAM Format Specification Working Group. Sequence Alignment/Map Format Specification. 2010; Available from: http://samtools.github.io/hts-specs/SAMv1.pdf

107. Wysoker A, Fennell T, Marth G, Abecasis G, Ruan J, et al. The Sequence Alignment/Map format and SAMtools. Bioinformatics. 2009;25(16):2078–9.

108. Samtools - Documentation (HTSlib manual pages). Available from: http://www.htslib.org/doc/

109. Zhan X. LiftOver. In: Genome Analysis Wiki [Internet]. 2015. Available from: https://genome.sph.umich.edu/wiki/LiftOver

110. Gao B, Huang Q, Baudis M. segment_liftover : a Python tool to convert segments between genome assemblies. F1000Research. 2018;7:319.

111. Li H. The missing human sequences (version 5). 2011; Available from: http://lh3lh3.users.sourceforge.net/download/decoyseq.pdf

112. Minikel EV. The decoy genome. 2013; Available from: http://www.cureffi.org/2013/02/01/the-decoy-genome/

113. FTP locations of the GIAB samples [Internet]. Available from: ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG002_NA24385_son/OsloUniversityHospital_Exome/
ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG003_NA24149_father/OsloUniversityHospital_Exome/
ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG004_NA24143_mother/OsloUniversityHospital_Exome/
ftp://ftp-

trace.ncbi.nlm.nih.gov/giab/ftp/data/ChineseTrio/HG005_NA24631_son/OsloUniversityHospital_Exome/

114. Platinum Genomes [Internet]. Illumina. Available from: https://www.illumina.com/platinumgenomes.html

115. Illumina Platinum Genomes [Internet]. Cloud Genomics | Google Cloud. Available from: https://cloud.google.com/genomics/docs/public-datasets/illumina-platinum-genomes

116. McKusick-Nathans Institute of Genetic Medicine, Johns Hopkins University (Baltimore M. OMIM - Online Mendelian Inheritance in Man [Internet]. Available from: https://omim.org/

117. Zacharia LF, Daly MJ, Gibbs RA, Kanani A, Cunningham F, Fukushima Y, et al. The International HapMap Project. Nature. 2003;426(6968):789–96.

118. Campbell CL, Scheller C, Horn H, Kidd JM, Doddapaneni H, Underhill PA, et al. A global reference for human genetic variation. Nature. 2015;526(7571):68–74.

119. Andrews S. FastQC: A quality control tool for high throughput sequence data [Internet]. Babraham Bioinformatics. 2010. Available from: http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

120. Broad Institute. Picard Tools [Internet]. 2009. Available from: http://broadinstitute.github.io/picard/

121. Li H. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. 2013; Available from: http://arxiv.org/abs/1303.3997

122. Christos H. Papadimitriou. Computational Complexity. Pearson; 1993.

123. Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to Algorithms. The MIT Press; 2009.

124. Zadeh R. The hard thing about deep learning [Internet]. O'Reilly Media. 2016. Available from: https://www.oreilly.com/ideas/the-hard-thing-about-deep-learning

125. Wang Y, Xu C, You S, Tao D, Xu C. CNNpack: Packing Convolutional Neural Networks in the Frequency Domain. In: Lee DD, Sugiyama M, Luxburg U V, Guyon I, Garnett R, editors. Advances in Neural Information Processing Systems 29 [Internet]. Curran Associates, Inc.; 2016. p. 253–61. Available from: http://papers.nips.cc/paper/6390-cnnpack-packing-convolutional-neural-networks-in-the-frequency-domain.pdf

126. Han S, Pool J, Tran J, Dally WJ. Learning both Weights and Connections for Efficient Neural Networks. 2015; Available from: http://arxiv.org/abs/1506.02626

127. Han S, Mao H, Dally WJ. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. 2015; Available from: http://arxiv.org/abs/1510.00149

128. Aghasi A, Abdi A, Nguyen N, Romberg J. Net-Trim: Convex Pruning of Deep Neural Networks with Performance Guarantee. 2016; Available from: http://arxiv.org/abs/1611.05162

129. Zook J. GitHub - Benchmarking Performance Metrics Definitions for SNVs and Small Indels [Internet]. 2015. Available from: https://github.com/ga4gh/benchmarking-tools/blob/master/doc/standards/GA4GHBenchmarkingPerformanceMetricsDefinitions.md

130. Cleary JG, Braithwaite R, Gaastra K, Hilbush BS, Inglis S, Irvine SA, et al. Comparing Variant Call Files for Performance Benchmarking of Next-Generation Sequencing Variant Calling Pipelines. bioRxiv [Internet]. 2015;023754. Available from: http://biorxiv.org/content/early/2015/08/03/023754.abstract

131. Olson ND, Lund SP, Colman RE, Foster JT, Sahl JW, Schupp JM, et al. Best practices for evaluating single nucleotide variant calling methods for microbial genomics. Front Genet. 2015;6(JUL).

132. GitHub - Illumina/happyR: R tools to interact with hap.py output [Internet]. Available from: https://github.com/Illumina/happyR

133. GitHub - aquaskyline/Clairvoyante: Clairvoyante: a multi-task convolutional deep neural network for variant calling in Single Molecule Sequencing [Internet]. Available from: https://github.com/aquaskyline/Clairvoyante

134. Tange O. GNU Parallel 2018. 2018; Available from: https://zenodo.org/record/1146014#.WrT_U9Yh0xM

135. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems [Internet]. 2015. Available from: https://www.tensorflow.org/

136. PyPy documentation [Internet]. Available from: http://doc.pypy.org/en/latest/

137. Garrison E. GitHub - vcflib/vcflib: a simple C++ library for parsing and manipulating VCF files [Internet]. Available from: https://github.com/vcflib/vcflib

138. Li H. A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. Bioinformatics. 2011;27(21):2987–93.

139. Chollet F, others. Keras [Internet]. 2015. Available from: https://keras.io

140. Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. arXiv e-prints [Internet]. 2016;abs/1605.0. Available from: http://arxiv.org/abs/1605.02688

141. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine Learning in Python. J Mach Learn Res. 2011;12:2825–30.

142. Jones E, Oliphant T, Peterson P, others. SciPy: Open source scientific tools for Python [Internet]. Available from: http://www.scipy.org/

143. Hunter JD. Matplotlib: A 2D graphics environment. Comput Sci Eng. 2007;

144. Cock PJA, Antao T, Chang JT, Chapman BA, Cox CJ, Dalke A, et al. Biopython: Freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics. 2009;25(11):1422–3.

145. Casbon, J. GitHub - jamescasbon/PyVCF: A Variant Call Format reader for Python [Internet]. Available from: https://github.com/jamescasbon/PyVCF

146. Jacobs, K. GitHub - pysam-developers/pysam: a Python module for reading and manipulating SAM/BAM/VCF/BCF files [Internet]. Available from: https://github.com/pysam-developers/pysam

147. GitHub - gedoardo83/GARFIELD-NGS: GARFIELD-NGS: Genomic vARiants FIltering by dEep Learning moDels in NGS [Internet]. Available from: https://github.com/gedoardo83/GARFIELD-NGS

148. GitHub - google/deepvariant: DeepVariant [Internet]. Available from: https://github.com/google/deepvariant/tree/r0.7

149. GitHub - google/nucleus: Python and C++ code for reading and writing genomics data [Internet]. Available from: https://github.com/google/nucleus

150. GitHub - CampagneLaboratory/variationanalysis at r1.4.0. Available from: https://github.com/CampagneLaboratory/variationanalysis/tree/r1.4.0

151. Campagne, F., Dorff, K. C., Chambwe, N., Robinson, J. T., & Mesirov, J. P. (2013). Compression of structured high-throughput sequencing data. PLoS ONE, 8(11). https://doi.org/10.1371/journal.pone.0079871

152. Chambwe N, Shaknovich R, Zeno Z, Campagne F, Simi M, Dorff KC. GobyWeb: Simplified Management and Analysis of Gene Expression and DNA Methylation Sequencing Data. PLoS One. 2013;8(7):e69666.

153. [CNNScoreVariants] A timeout ocurred waiting for output from the remote Python command · Issue #4696. (2018) [Internet]. Available from: https://github.com/broadinstitute/gatk/issues/4696

154. Timeout Error while using CNNScoreVariants — GATK-Forum. (2018) [Internet]. Available from: https://gatkforums.broadinstitute.org/gatk/discussion/12263/timeout-error-while-using-cnnscorevariants

155. Roy S, LaFramboise WA, Nikiforov YE, Nikiforova MN, Routbort MJ, Pfeifer J, et al. Next-generation sequencing informatics: Challenges and strategies for implementation in a clinical environment. Arch Pathol Lab Med. 2016;140(9):958–75.

Appendices

Supplementary table 1.

Source	NA12878 (HG001)	NA24385_son (HG002)	AshkenazimTrio NA24149_father (HG003)	NA24143_mother (HG004)	NA24631_son (HG005)	ChineseTrio NA24694_father (HG006)	NA24695_mother (HG007)
10XGenomics	WGS/LR, 10x Gen.	WGS/LR, 10x Gen.	WGS/LR, 10x Gen.	WGS/LR, 10x Gen.			
10Xgenomics_ChromiumGenome	WGS/LR, 10x Gen.	WGS/LR, 10x Gen.	WGS/LR, 10x Gen.	WGS/LR, 10x Gen.			
BGISEQ500	WGS, BGISEQ						
BGISEQ500 stLFR	WGS, BGISEQ						
CompleteGenomics_normal	WGS/LFR, Complete Gen.						
CompleteGenomics_normal_RMDNA	WGS/PE, Complete Gen.	WGS/PE, Complete Gen.	WGS/PE, Complete Gen.	WGS/PE, Complete Gen.	WGS/PE, Complete Gen.		
CompleteGenomics_normal_cellsDNA					WGS/PE, Complete Gen.	WGS/PE, Complete Gen.	WGS/PE, Complete Gen.
Garvan_NA12878_HG001_HiSeq_Exome	WES/PE, Illumina						
MtSinai_BioNano	WGS/OM, BioNano	WGS/OM, BioNano	WGS/OM, BioNano	WGS/OM, BioNano	WGS/OM, BioNano		
NA12878_PacBio_MtSinai	WGS/SE, Illumina	WGS/SE, Illumina	WGS/SE, Illumina	WGS/SE, Illumina	WGS/SE, Illumina		
HiSeq_300x	WGS/PE, Illumina	WGS/PE, Illumina	WGS/PE, Illumina	WGS/PE, Illumina	WGS/PE, Illumina		
HiSeq100x						WGS/PE, Illumina	WGS/PE, Illumina
NIST_NA12878_HG001_SOLiD5500W	WGS/SE, SOLiD	WGS/SE, SOLiD			WGS/SE, SOLiD		
Nebraska_NA12878_HG001_TruSeq_Exome	WES/PE						
ion_exome	WES, ION	WES, ION	WES, ION	WES, ION	WES, ION		
CORNELL_Oxford_Nanopore		WGS/2D, ONT					
Ultralong_OxfordNanopore		WGS/1D, ONT					
Dovetail_ChicagoLibraties		WGS/PE, Illumina/PacBio	WGS/PE, Illumina/PacBio	WGS/PE, Illumina/PacBio			
NIST_Illumina_2x250bps		WGS/PE, Illumina	WGS/PE, Illumina	WGS/PE, Illumina			
NIST_Stanford_Illumina_6kb_matepair		WGS/PE, Illumina	WGS/PE, Illumina	WGS/PE, Illumina	WGS/PE, Illumina	WGS/PE, Illumina	WGS/PE, Illumina
NIST_Stanford_Moleculo		WGS/PE, Illumina	WGS/PE, Illumina	WGS/PE, Illumina	WGS/PE, Illumina	WGS/PE, Illumina	WGS/PE, Illumina
OsloUniversityHospital_Exome		WES/PE, Illumina	WES/PE, Illumina	WES/PE, Illumina	WES/PE, Illumina		
PrecisionFDA	WGS/PE, Illumina	WGS/PE, Illumina					
Platinum Genomes	WGS/PE, Illumina, pedigree						

Supplementary table 2.

HG002	HG003	HG004	HG005
1.0 (~135x cov)	1.0 (~116.6x)	1.0 (~128x)	1.0 (~109.5x)
0.9 (~121.5x)	0.9 (~105x)	0.9 (~115.3x)	0.9 (~98.5x)
0.8 (~108x)	0.8 (~93.3x)	0.8 (~102.5x)	0.8 (~87.6x)
0.7 (~94.5x)	0.7 (~81.6x)	0.7 (~90x)	0.7 (~76.7x)
0.6 (~81x)	0.6 (~70x)	0.6 (~76.9x)	0.6 (~65.7x)
0.5 (~67.5x)	0.5 (~58.3x)	0.5 (~64x)	0.5 (~54.8x)
0.4 (~54x)	0.4 (~46.6x)	0.4 (~51.2x)	0.4 (~43.8x)
0.3 (~40.5x)	0.3 (~35x)	0.3 (~38.4x)	0.3 (~32.8x)
0.2 (~27x)	0.2 (~23.3x)	0.2 (~25.6x)	0.2 (~21.9x)
0.1 (~13.5x)	0.1 (~11.7x)	0.1 (~12.8x)	0.1 (~11x)