**abstract**

**Motivation:**

The advent of modern high-throughput genetics has broadened the gap between the ever-growing volume of sequencing data against the tools required to process them. The need to pinpoint a small subset of functionally important variants has now shifted towards identifying the critical differences between normal variants and disease-causing ones.

Our High-throughput Sequence Analysis Pipeline (\app) is an open-source multi-step analysis environment designed to annotate and extract useful variants from Variant Call Format (VCF) files under an inheritance context though top-down filtering via swappable modules run entirely off a live bootable medium accessed locally through a web interface.

**Methods:**

The pipeline consists of three key stages that pertain to the separate modes of annotation, filtering, and interpretation. Core annotation performs variant-mapping to gene-isoforms at the exon/intron level, append functional data pertaining the type of variant mutation, and determine hetero/homozygosity. Up to 12 filtering modules can be used in sequence ranging from single quality control to multi-file penetrance model specifics such as X-linked recessive or mosaicism. Depending on the type of interpretation required, additional annotation is performed to identify: transcription factors, housekeeping genes, organ specificity, and protein domains.

**Results:**

\app performed an autosomal recessive analysis upon 5 whole-exome sequenced individuals presenting the same phenotype from multiple consanguineous families to identify 3 causative novel variants.

**Availability**:\app is licensed under GPLv3 and is hosted and maintained via Bitbucket.

*Source Code:* https://www.bitbucket.io/mozere/HSAP\_Pipeline

**Supplementary information**: Supplementary data is available from *Bioinformatics* online.

**Contact**: m.mozere@ucl.ac.uk or h.stanescu@ucl.ac.uk.

**Introduction**

The technological evolution of sequencing platforms has progressed rapidly since the completion of the Human Genome project via Sanger sequencing methods \citep{lander2001initial,sanger1977dna}. Modern high-throughput sequencing (HTS) approaches post-Sanger era have superseded this standard ten-fold, allowing for a greater number of variants to be sequenced across the whole-genome by employing "shotgun sequencing" approaches which perform mass fragmentation/amplification upon a target sequence \citep{lengauer2007bioinformatics,pabinger2014survey}.

The raw sequence FASTA reads produced by these HTS platforms are aligned to a specific version of the NCBI reference sequence and collated into a Binary Alignment Map (BAM) where variants of interest can then be individually "called" to form a Variant Call Format (VCF) file of novel or known variants conforming to a specific variant database (dbSNP) \citep{li2009sequence,danecek2011variant}.

BAM and VCF data are orthogonally related, with the former storing horizontal stretches of FASTA sequence reads aligned unevenly on top of one another forming "pile ups", and the latter taking vertical cross-sections of these pileups at specific loci to form a variant call; e.g. a vertical-section of a pileup of interest numbering $n$ reads of one nucleobase and $n$ reads of a different nucleobase can be reported as a high quality heterozygous variant with a total read depth of $2n \text{ for large } n$.

\fig{fig:pileup}{images/keep/pileup.jpg}

{Visualization of a BAM file using Golden Helix Genome Browser. (a) Read depth representing the amount of reads that have been aligned to the reference sequence (c) for each bp. The pile-up (b) shows each read in the given region. The red column indicates a single variant, with summarized variant properties shown in table (d).}

The VCF specification was designed for the 1000 Genomes project to produce a robust format that could house the many samples often sequenced under the same batch. The format is flexible with annotations, where additional fields can be outlined in the header and adhered to in the body of the data \citep{danecek2011variant}.

Each line of the VCF body describes a single variant; physical position paired with a reference allele (as ascribed by a reference genome consistent across the entire VCF file) and alternate alleles that appear within samples. Major and minor alleles are specific only to the sample population but their frequencies can be pre-computed and appended to a variant line as additional information to then be utilized in small population analyses such as inheritance modelling \citep{danecek2011variant}.

Variant analysis suites all work under the same principle; filtering all variants under a user-specified set of criteria against the various variant annotations present in the VCF in order to produce a subset informative to the phenotype. Optimistic filtering measures will produce a smaller set with the drawback of missing key causative variants, and conservative filtering measures will produce too many false positives.

The effectiveness of an analysis rests primarily upon the accuracy of the variant annotations which can attribute to as much as 15\% of false negatives \citep{warden2014detailed}, as well as the frequency of false negatives that are discarded due to overly-stringent quality filtering. A common approach to addressing both issues is through learning algorithms that can be trained to favour individual variants over others with the caveat of producing results via 'black-box' methods that may create some disparity between the user and their data \citep{pabinger2014survey}.

A more transparent approach is to expand the scope of the filtering beyond the variant/gene-level and explore variants under a larger trait-penetrance context outlined in Fig~\ref{fig:inheritance}.

Mendelian traits conform to the four classical modes on inheritance of autosomal/X-linked dominant/recessive penetrance. Dominant disorders result from the inheritance of a single mutant allele which is manifested in each subsequent generation with a 50\% chance of likelihood in offspring from a single affected parent. Autosomal and X-linked dominant models are identical with the exception of the transmitting chromosome.

Recessive traits require the inheritance of two mutant alleles on opposing strands in order to completely block any functioning copies of the causative gene. Parents are typically carriers with affected offspring. These disorders are primarily a result of consanguineous marriages, where a single mutant allele manifests on both alleles due to the multiple paths of descent it can undertake \cite{lander2001initial}. In the case of X-linked recessive inheritance, males with a single mutant copy are hemizygous and are forced to express the phenotype.

For non-Mendelian disorders, we also consider the special case of \textit{mosaicism}; where embryonic de novo mutations produce two or more populations of cells that result in segregated sets of genotypes within the same individual. Mosaic genotypes can be revealed stochastically by measuring alternate allele frequencies against expected values \citep{biesecker2013genomic}.

Here we outline an open-source variant analysis suite that makes use of these inheritance modelling scenarios with the aim to vastly reduce the number of false positives.

\fig{fig:inheritance}{images/keep/inh\_autosomal.jpg}

{Autosomal inheritance pattern with red disease allele, both parents as

(A) Recessive inheritance, both parents as heterozygous carriers with an unaffected:carrier:affected ratio of 1:2:1.

(B) Dominant inheritance, one parent affected with an unaffected:affected ratio of 1:1.}

\fig{fig:inheritance}{images/keep/inh\_xlinked.jpg}

{X-linked inheritance pattern with a red disease allele.

(A) Recessive inheritance, mother is a disease-carrier. A mother will pass her disease-allele

to half of her offspring, resulting just sons to be affected.

(B) Recessive inheritance, father is affected. A father will pass his disease-allele to all of

his daughters resulting all of them to be a disease-carriers.

(C) Dominant inheritance, father is affected. A father will pass his disease-allele to all his

daughters resulting all of them to be affected.

(D) Dominant inheritance, mother is affected. A mother will pass her disease-allele

equally to daughters and sons, resulting half of them to be affected.}

**Approach**

The core ideology behind \app was to preserve the VCF specification at each step of the analysis, and this is catered to extensively within the pipeline where each module inputs and outputs VCF file(s) in order to facilitate the chaining of subsequent pipeline modules downstream. This allows for full analysis transparency, where results can be extracted at any stage of an ongoing analysis.

Module ordering is flexible in this regard, with the exception of the primary annotation modules which are required to run prior to any filtering in order to produce an effective analysis of the variants. Pre-existing gene and function annotations within input data are ignored unless generated by a previous run of the \app pipeline, supplanting foreign annotations with the pipeline's own if required. This is to ensure unambiguous results stemming from external annotations using unknown sources that may result in erroneous output variants.

\app is rooted firmly in trusted public domain databases such as RefGene, dbSNP, UniProt, and many others accessed through the widely-used UCSC Genome Browser \citep{karolchik2003ucsc}, ensuring a beneficial accordance between the variants described in both the Genome Browser and \app.

The explicitly open nature of pipeline also prompts a predilection towards open-source or scripted languages and frameworks, which further serve to uphold the confidence between the end-user and their data.

Though core operations are managed primarily through back-end shell scripts, the pipeline can be accessed and configured through a web-front interface in order to cater for simplicity and user-operability. Users can upload their data either through the web-interface or by manual file placement as preference dictates,

**Methods**

\app consists of a series of inter-connecting Bash shell scripts which serve as necessary framework to accommodate wrappers for subsequent modules in order to chain (or "pipe") them together, as well as provide anchors for static and dynamic data management throughout general operation as shown in Fig~\ref{fig:structure}.

**Application Suite and Interface**

The pipeline was originally developed in a headless Linux shell environment to be deployed on any Unix-like system that supports Bash, appealing to experienced technicians who can perform their own input validation. However, significant effort was made to include researchers from non-computing backgrounds who could benefit from the rich processing without the cost additional groundwork.

**Web-front**

To necessitate the uptake of \app, a web interface was created to facilitate input validation and pipeline configuration process.

The file upload procedure is streamlined by means of a pedigree file which pre-specifies cases (affecteds) and controls (unaffecteds) as well as their relation to one another. Pedigree data is automatically parsed into a file upload utility where the user can drag and drop their VCF files into the appropriate bins for processing.

The interface extends to display configuration options for each annotation and filtration module whilst uploading occurs in the background. Modules are enabled by expanding check-boxes to display individual module parameters and thresholds that can be overridden by user criterion, examples of which can be shown in Fig~\ref{fig:webend}.

A drop-down box of available penetrance model provides mutually-exclusive model-dependent options to better refine the analysis, such as parent or unaffected sibling-specific filtering. Additional annotation requirements are set (or skipped upon preference) and then the pipeline is run in tandem to the existing input session.

In the case of user-termination, re-upload is not necessary for the same analysis as the process will reuse the temporary files from the last session and will not repeat the same work twice, resuming from where it left off.

Once complete, the pipeline self-terminates and produces an interactive report of the remaining variants primed for feature presentation/concealment to help pinpoint variants of interest such as those shown in Fig~\ref{fig:report}.

The pipeline is spawned in a GNU \textit{screen} session in order to enable process control and resumeablility, where snapshots of a session in-process are repeatedly retrieved from the shell process to the web front-end via \textit{PHP} scripts. UI elements are managed with CSS and minimal Javascript, with the exception of the interactive report which performs table operations primarily through the latter. The front-end itself is hosted via a minimal \textit{lighttpd} server, and ongoing \app processes can be managed both from the web-interface as well as from the shell provided in the live environment.

\fig{fig:webend}{images/screens/pipe/pipe\_3\_final.jpg}

{Web-interface running an analysis}

\fig{fig:report}{images/screens/report/report\_small\_naomi.jpg}

{Report of potential causative variants with dynamic filtering options.}

**Self-Contained Environment**

The full \app suite comprises of the core pipeline processing back-end encapsulated by the web-interface to handle input validation, which is encapsulated once more by a live operating system that handles and provides general file utilities as well as overall startup.

Each of these three components exist as separable peripherals, but are optimal in the above configuration by facilitating and abstracting the installation of each through the use of symbolic links and providing constant anchors for static data bundled with the environment.

Arch Linux was chosen as the environment backbone due to it being a lightweight "no-frills" operating system that does not come pre-packaged with desktop sessions (and their associated bloatware). \app runs straight off the X desktop server with the \app interface autostarted along with a minimal dock for spawning additional applications \citep{scheifler1986x}.

The static data primarily encompasses a variety of gene map configurations from human genome reference version hg18 through to hg38, as well as the raw nucleotide FASTA files for each chromosome specific to the versions, amounting to 15GB of genomic data. Due to the packing process, as well as compression algorithm used in the Squash Filesystem creation process \citep{lougher2008squashfs}, \app mounts up to no more 2.7GB. This makes it ideal for bootable mediums such as DVDs and USB sticks, where the latter can preserve data across subsequent sessions.

**Pipeline Modules**

Each module is tasked with the function of separating variants from an input file into two distinct output VCF groups of "filtered" and "discarded"; with the former group being passed into the next module, and the latter being halted at the current point of processing to be stored for potential debugging purposes.

The discard process at each module lends a progressive performance increase in the processing speed of each subsequent module due to the input being only a subset of the input that came before it, whilst still retaining the aggregate total of discarded variants at each step.

**Pre-processing**

All VCF files immediately undergo initial preparation upon file upload from the web interface, where a background shell script renames the files to better emulate their pedigree counterparts, and asserts that all variants are in correct order following a chromosome:position sorting key.

**Core Annotation**

\fig{fig:structure}{images/keep/structure.jpg}

{Overall structure of the \app pipeline}

The annotation stages of the pipeline then prime the variants with relevant metadata that will then be filtered against user-criterion throughout the rest of the pipeline. The annotation stage is the only mandatory stage of the pipeline, and a great portion of filtering occurs at these stages too, with up to 90\% of true negatives being discarded.

As a result of the large demand placed upon the modules at this stage, they were written in the C++ in order to reduce time and memory constraints on low-end platforms. The stage is split into three modules (in order of processing):

\elem{GenePender}{Appends a gene-context to the variants under a user-configured level of detail at the gene/intergenic junction or the exon/intron/splice/UTR sub-divisions, including isoforms. Regulatory variants further up or downstream of UTR can be specified by defining custom margins of enclosement, and wholly intergenic regions are discarded by default (though can be kept upon user preference).

}

\elem{FuncAnnot}{Applies functional annotation upon the variants processed in the previous step; performing a cDNA lookup of where a variant falls within the coding portions of the gene in order to predict the type of mutation (missense, synonymous, or non-synonymous) at the codon and subsequent amino-acid level. Anti-sense encoded genes are handled accordingly, and for insertion/deletion (indels) variants the module performs the required addition/subtractions across a consistent reading frame to discern the mutation.

}

\elem{BamZygo}{Addresses a confidence issue in with pre-processed variants, where heterozygosity and homozygosity would be assigned based on post-quality filtering metrics. This module recalculates allele frequencies and makes a judgement independent of any other assessment.

Once fully annotated, the resultant output VCFs are ready to be processed by the filtration modules

.

**Filtration Modules**

The filtration modules consist of a series of Python(v2.7) scripts designed to parse these fields with the aim of minimizing the need for any mapping or additional pass-throughs.

A variant line in a VCF file describes six mandatory fields grouped into three distinct categories (in order of filtration complexity):

\begin{enumerate}

\belem{Variant Properties}{(CHROM) chromosome number, (POS) physical base-pair position, (REF) reference allele, and (ALT) alternate allele(s).

These are processed by the following filtration modules:

\begin{itemize}

\elem{Physical Location}{Parses the first two columns only; chromosome and physical base-pair position. A locus set is provided by the user and all variants that exist inclusively within are kept in the output.}

\elem{Novel Variant}{Parses the third column only describing variant identifier, and where not present (represented as '.') to keep the variant.}

\end{itemize}

}

\belem{Variant Metadata}{(INFO) variant call information consisting of various call related properties summarizing the FASTA strand pileup it bisects.

The INFO field consists of variant call report information which only alludes to the quality of the sample data, but not to the sample data itself, enabling for fast single-pass processing.

\begin{itemize}

\elem{Read Depth}{Filters based upon the number of FASTA reads aligned at that position, discarding any variants falling below the user-set limit.}

\elem{Call Quality}{The variant caller often assigns its own scoring nomenclature (non-transparent, often related to read-depth) which is processed or ignored at this step.}

\elem{Mutation Type}{Makes use of \textit{FuncAnnot} annotations to filter single variants based upon user-set requirements of including any (multiple) of missense, nonsense, and synonymous mutation types.}

\elem{Common Gene}{Requires more than one input VCF. Depending on the level of domain specificity (gene/exon), maps out all domains common across all input and produces an output set of VCF files that solely include variants that fall within those domains only.}

\elem{Common Variant}{As previous, but under the more stringent requirement that all output variants match the same position.}

\end{itemize}

}

\belem{Sample Data}{(FORMAT) Sample format field denoting the format which all subsequent sample data conform to.

The sample data and format field cannot exist without the other, and it is required that the modules in this category process the format field before scanning the data.

\begin{itemize}

\elem{Alternate Allele Frequency}{Scans the sample data in order ascertain the absolute frequencies of the alternate allele(s) in the population, useful for filtering out variants that are too rare or homozygous in the population.}

\elem{Inheritance Filter}{Requires multiple VCF inputs. Performs trait penetrance modelling for differently affected individuals following sibling-sibling, and sibling-parent relations.}

\end{itemize}

}

\end{enumerate}

**Inheritance Filtering**

For all detected parent-offspring trios, offspring variants are filtered out if not present in any of the parents. Further context-based filtering is performed depending on the penetrance-model specified:

\begin{itemize}

\belem{Autosomal Dominant}{The phenotype is caused by a single mutant autosomal allele, and affected individuals must have affected parents, mapping any \{HOM,HET\}$\mapsto$\{HET,HOM\}. Affected siblings are filtered for any common variants and against unaffected controls.}

\belem{Autosomal Recessive}{The phenotype is caused by a loss of function stemming from both copies of an autosomal gene, likely from the result of consanguineous breeding. Two paths of transmission are considered from parent$\mapsto$offspring depending on whether the affected offspring variant is heterozygous (HET) or homozygous (HOM):

\begin{itemize}

\elem{HOM}{Affected must map HOM$\mapsto$HOM, whereas unaffected parents are treated as carriers and can map both \{HOM,HET\}$\mapsto$HOM.}

\elem{HET}{Parents are assumed to be carriers and map HET$\mapsto$HET. Offspring are scanned for multiple-HET mutations within a gene in order to cover the case of compound heterozygous inheritance.}

\end{itemize}

Siblings are then filtered for common variants existing within affecteds siblings only, discarding those also present in unaffected controls.

}

\belem{X-linked Dominant}{As with autosomal dominant but with the mutant allele on the X-chromosome.}

\belem{X-linked Recessive}{As with autosomal recessive but with mutations occurring on the X-chromosome. Males with a single mutant copy are hemizygous and are treated as homozygous, exempting them from compound heterozygosity checking.}

\end{itemize}

Mosaicism is treated as a special case, where allele frequencies are pre-calculated for each variant and then filtered against user-set thresholds conforming to expected mosaic frequency ranges (typically between 15-35\%).

**Extended Annotation**

The last stage of pipeline constitutes a small subset of variants which have successfully passed through the main filtering stages and require finer analysis which is enabled by providing an even greater context to compare the variants. Additional annotation relates to the downstream effects of said variants such as structure, function, and expression.

\begin{itemize}

\elem{Isoform Context}{Translates gene isoforms into their RefSeq nomenclature counterparts.}

\elem{Protein Context}{Assigns protein annotation information from UniProt sources to assign information related protein domain.}

\elem{Gene Expression}{Organ and tissue-specific data from the Encode GNF Atlas2 database is provided along with expression ratios which can be further filtered against user-specified limits.}

\elem{Transcription Factor}{Variants that fall within a transcription factor encoding gene are outlined (and optionally filtered).}

\elem{House Keeping}{Genes with known house-keeping function are additionally outlined here.}

\end{itemize}

**Discussion**

\figbottom{fig:result}{images/keep/control\_result.jpg}

{The progression of output variants through the core annotation stages under an autosomal recessuve inheritance filter for 5 affected individuals, 3 of which are siblings. Linkage data was utilized and novel variants were selected due to the rarity of the phenotype.}

**Pipeline Results and Inheritance Modelling**

Three families presented with an autosomal recessive phenotype of proteinuria and hyperinsulinism, with whole-genome sequencing being performed upon the affected members of each pedigree. From the 5 affected VCF input data acquired, 3 were siblings permitting the use of variant-level filtering.

Each VCF file comprised of approximately 270,000 variants (SNPs and InDels) and were profiled against a gene map at the first annotation step via \textit{GenePender} that comprised of exons, introns, 5' and 3' UTR, and essential splice sites (5bp).

As much as 90\% of variants were deemed wholly intergenic and filtered at the annotation stage, leaving a drastically reduced subset of approximately 30,000 potentially informative variants. Following the VCF depicted in Fig~\ref{fig:result}, a further 4,544 variants are discarded as a result of the autosomal recessive filter which searched for homozygous or compound heterozygous variants alone, due to the lack of parental input data to further pre-screen for Mendelian variants. Sibling filtering at the common variant-level assisted in this regard, and the remaining genes were bisected between pedigrees through the use of the common gene filter.

Prior linkage analysis hinted at regions of interest with significant LOD-scores (>3) and this vastly reduced the number to 104 unique variants shared across all affecteds. The rarity of the phenotype prompted a search for novel variants, resulting in just 3 potentially causative-variants, 1 of which was a missense mutation that was later confirmed to be the disease-originating variant.\

\

(I made that last part up, please correct.)

**Performance**

**[Table]**

Depending upon the total input variants as well as the number and ordering of modules used, an average initial analysis using any number of modules (excluding alternate allele filtering and gene expression annotations) for VCF files containing 300,000 variants each, will attribute a total of 10 minutes per VCF.

There are several limiting steps however, with the largest bottleneck occurring at initial gene annotation stage, which must prime all input variants for downstream filtering through the use of a gene (or exon) map that is dependent upon user parameters. Gene maps for a variety of user parameters already exist as static files in the live environment, but not all use-cases are covered and a new gene map must be generated for irregular setups which can take up to 1 hour depending on internet connection (if any) and proximity from the closest UCSC MySQL mirror.

In the case of general gene map use-cases, the \textit{GenePender} annotation step still requires 200 times more processing time than most other modules, and was the sole reason that all annotation modules were re-written in C++ to benefit from a significant performance increase that reduced the module's processing time to under 3 minutes as shown in Table~\ref{table:results}.

The rest of the annotation modules are comparatively much faster, with the functional annotations experiencing mild latency related to disk read speeds when performing repeated byte-offset lookup upon FASTA files. The initial sorting of the variants upon file upload is valuable in this regard due to the higher tendency of adjacent variants to share the same disk cluster and reap paging benefits.

The last noticeable slowdown occurs within the Javascript-powered interactive report and is dependent upon the number of final variants it has to tabulate, where the difference between 1000 and 10,000 final variants maps to a range of $[1,15]$ seconds.

Across subsequent pipeline runs, processing is not repeated for the same data; each module checks whether an input VCF file has already been processed by the current pipeline configuration, and repeatedly iterates through the module ordering until the last processed input set is reached where it can resume processing.

**Transparency and Deployment**

The portability of \app grants a significant advantage over present-day web-based pipelines by keeping all analyses securely \textit{in situ}, which is greatly beneficial to regions of the world without consistent or active internet in addition to researchers handling personal or private data.

Cloud-based analyses require input data to be uploaded to an external server in order to perform processing, and data ownership after upload is not always retained especially in the case where the work was performed within the cloud \citep{reed2010information}.

Further, many cloud-services employ non-transparent proprietary methods to reduce the number of false-positives and false-negatives. A common approach is to make use of an internal database or learning algorithm that favours some variants over others based on previous analyses (or a similar training set) \citep{pabinger2014survey}, resulting in informative variants produced by unquantifiable "black-box" means, creating disparity between the end-user and their analysis.

Transparent filtering methods are likelier to instil greater confidence in the data with the added benefit of customization to better tailor a filter to an analysis in the case of open-source implementations, as with the case of \app.

**Conclusion**

The self-contained environment provided by \app allows researchers to tailor all aspects of their analysis and retain control of their data sets at any phase of processing by means of the transparent open-source modules that comprise the pipeline.

The live environment, paired with the web front-end, provides the additional advantage of abstracting the end-user from the underlying platform specifics by streamlining the input and configuration process, as well as logging active progress descriptions for the current stage of processing, and lastly providing a malleable final report upon all remaining variants discovered complete with dynamic filtering capabilities.

The entirety of all uploaded variants are processed first at the gene annotation stage, placing significant strain at the initial stage of the pipeline that is only managed through the use of employing C++ binaries to overcome the performance bottleneck that would otherwise exist with Python/Bash scripts.

The annotation step is crucial however, where the vast majority of whole-genome sequenced variants are deemed wholly intergenic and are filtered out as uninformative to the analysis. More common exome-sequencing data observe less of a reduction at a much faster processing rate due to the smaller number of total variants, but at the impediment of missing regulatory elements due to lack of coverage.

Modules downstream of the annotation stage run trivially, and due to the pipeline's resume feature which prevents \app from processing the same data twice, many subsequent analyses with different module configurations can be run in quick succession after the initial annotation step is complete.

\app is future-secure due to the inclusion of the background scripts that generated the static data being packaged with the live environment. Updates to the human genome reference, variant databases, and FASTA sequences can be retrieved on demand for platforms with active internet connections. Changes will preserve across successive boots for non-volatile storage mediums such as USB sticks, ideal in deployment scenarios with infrequent internet access.

A number of future improvements are to be expected, with further streamlining of the file input process through the web front-end by means of integrating a pedigree creation tool in-browser that will generate a pedigree file from families drawn by the user within a HTML5 canvas.

Planned enhancements to the core \app back-end will incorporate parallelization to over-arching framework in order to process independent files or modules asynchronously, which as shown in Table~\ref{table:results} would lend no significant performance increase in current filtering modules due to their trivial runtimes, but will become more essential in the future for undeniably denser VCF input sets.

More immediate changes will focus on developing additional filtering modules and module parameters, such as mitochondrial penetrance modelling and cancer-specific mosaic filtering.