Investigating Bioinformatic Pipelines for De novo Variant Identification in New Zealand Dairy Cattle

A thesis

Submitted in partial fulfilment

of the requirements for the degree

of

Master Of Science (Research) in Molecular and Cellular Biology

at

The University of Waikato

by

Erika Duerre

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2023

Abstract

New Zealand is the biggest single entity for milk and milk product export in the world and being the country's main industry, keeping track of bovine mutations allows influence of the national herd's health and the ability to breed cattle better suited to the current farming and climate needs. With advances in next-generation sequencing of whole genomes, research understanding and tracking germline de novo mutations (DNMs) is attainable. However, current techniques being used to identify and study candidate DNMs are unvetted and experimental. Some of these techniques include bioinformatic software packages such as DenovoCNN, DenovoGear, PedFilter, and RUFUS.

This thesis aimed to investigate and test two of these identification pipelines, DenovoGear and PedFilter using next-generation sequencing data from bovine trios. Output data was generated by writing scripts and extracting data from six trios into Excel workbooks or manual IGV validation of selected candidate DNMs. Of the total 714,393 candidates identified across PedFilter and DenovoGear, only 161 of them were identified by both programs. After Integrative Genomic Viewer validation, 50% of these candidates were deemed to be true positives while only 12% were deemed false positives. Data analysis shows neither technique is 100% accurate and they both should be used with secondary filtering and confirmation of the identified DNM candidates. It was also apparent that the different techniques would be better suited to different study situations. For example, (1) For smaller studies (e.g., 3–15 trios), with limited data available to them, using DenovoGear with a secondary software program for filtering would be best (2) If sufficient samples are available to provide a simulated population data set, PedFilter has higher accuracy than DenovoGear on its own (3) For fast analysis without requiring heavy filtering of the output data use a combination of both PedFilter and DenovoGear.

Lastly, it is clear that further developments and more testing is required in this area of research before a set standard of practice or guidelines can be recommended and implemented. It is suggested to testing DenovoCNN and Rufus or branching out to other packages to trial like SynthDNM, DNMFilter_INDELs or HAPDeNovo.

Acknowledgements

The production of this thesis is thanks to multiple people and organisations' support and help.

Firstly, I would like to sincerely thank my primary supervisor Dr. Chad Harland for all his experience and guidance throughout my project.

Next, I would like to express gratitude to Livestock Improvement Corporation (LIC) specifically the Research and Development department who supplied the genetic material for testing as well as their collective expertise in genetic and bioinformatic research.

Thirdly I would like to thank my two additional supervisors Dr. Linda Peters, of The University of Waikato and Dr. Thomas Lopdell of LIC.

I also extend my thanks to the University of Waikato for providing the opportunity to extend my studies while researching such a relevant topic.

Lastly, I would like to acknowledge my partner Joel Taylor, my family and his family for their continued support throughout.

Table of Contents

List of Figures

List of Equations

List of Tables

List of Abbreviations

C	Child of a trio
Chrom/Chr	Chromosome
INDEL	For mutations that are Insertions of Deletions
DNM	Denovo Mutations (for the purpose of this study Germline Denovo Mutations unless otherwise specified.
DNMsc	Denovo Mutation Score. The probability Scoring system for DenovoGear
F	Female mother of a trio
FP	False Positive
FN	False Negative
GO	Grand Offspring
GP	Grand Parent
HS	Half-Sibling
IGV	Integrative Genomic Viewer
LIC	Livestock Improvement Corporation
M	Male father of a trio
MGS	Maternal Grand-Sire

MGD	Maternal Grand-Dam
NGS	Next Generation Sequencing
PGS	Paternal Grand-Sire
PGD	Paternal Grand-Dam
Pos	Position, usually referring to the position on a chromosome
Pr	Proband
S	Sire
SNP	Single Nucleotide Polymorphism
TP	True Positive
TN	True Negative
WGS	Whole Genome Sequencing

Chapter 1: Literature Review

Introduction

De novo mutations (DNMs) are new spontaneous genetic changes in DNA such as a single nucleotide polymorphism (SNP), an insertion or a deletion (INDEL) that are present in an individual and absent in their biological parents (Acuna-Hidalgo et al., 2016; Goldmann et al., 2019; Veltman & Brunner, 2012). This genetic variation is usually neutral, but sometimes can be deleterious, or in rare cases, beneficial. Cells can naturally accumulate DNMs through spontaneous mutagenesis during cell division. Thus, the identification of DNMs increases our understanding of evolution (Kong et al., 2012) as well as human health and disease (Acuna-Hidalgo et al., 2016; Gill et al., 2016; Goldmann et al., 2018, 2019; Kong et al., 2012; Ségurel et al., 2014; Veltman & Brunner, 2012).

In the last decade, it has become possible to study these de novo germline mutations through the ascertainment and sequencing of familial trios (Figure 1), the DNA testing of samples from a child and both biological parents, the rapid improvement of whole genome DNA sequencing technologies (Bergeron et al., 2022; Harland et al., 2017) and development of processes to analyse the resulting large data sets.

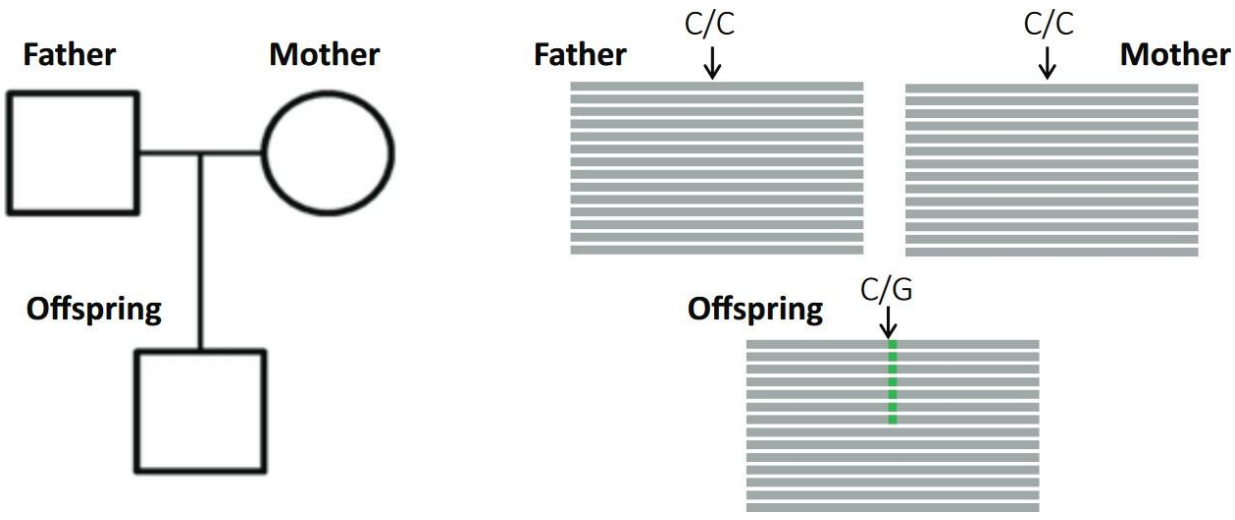


Figure 1: Detection of a De novo mutation in a Trio Sample (mother, father, and offspring). Potential candidates for DNMs are sites where approximately half of the reads (indicated as grey bars) from the offspring have a variant (indicated in green) that is absent from the parental reads.

While most studies focus on human DNM, research has expanded to include a range of other organisms such as domestic cattle, *Bos taurus* (Bergeron et al., 2022). For New Zealand, cattle are an essential part of the economy. New Zealand is the biggest single entity for milk and milk product export in the world at 30% of the free trade market (Jay, 2007). Being the

country's main industry, keeping track of bovine mutations allows influence of the national herd's health and the ability to breed cattle better suited to the current farming needs (Harris, 2005). It also allows for detecting and tracking deleterious mutations, such as the hairy/slick hair gene, to prevent them from entering or further propagating through the population (Littlejohn et al., 2014). This is important because deleterious mutations can reduce production as they can lead to animal health and welfare issues.

Estimation of De Novo Mutations Rates

Veltman & Brunner, 2012, stated that "*De novo mutations represent the most extreme form of rare genetic variation: they are more deleterious, on average than inherited variation because they have been subjected to less stringent evolutionary selection*" (Veltman & Brunner, 2012, p.565). This can be seen in some of the known diseases and disorders caused by or linked to de novo mutations (Acuna-Hidalgo et al., 2016; Goldmann et al., 2019; Kong et al., 2012). Early research following these deleterious mutations can be seen in Haldane's 1935 study following the occurrence of haemophilia in London England. In this study, DNM rates were estimated using the prevalence of haemophilia in the population (Acuna-Hidalgo et al., 2016; Haldane, 1935). At the time phenotypic expression of DNM was the only information available to track and study these spontaneous causes for some genetic disorders. With the advent of Sanger Sequencing (Men et al., 2008) and similar technologies in the 1990s single gene or region estimates for DNM genotypes were possible (Drake et al., 1998). Following Sanger Sequencing, the development and use of modern next-generation sequencing (NGS) techniques and bioinformatics have provided more in-depth research opportunities using whole genome sequencing, to isolate and better understand the role de novo mutations play.

For example, it was previously hypothesised that male sperm were constantly dividing and therefore had a large potential for de novo mutations (Heller & Clermont, 1963). This contributed to the idea that there is a correlation between the father's age at conception and the number of de novo mutations seen in offspring (Acuna-Hidalgo et al., 2016; Goldmann et al., 2019; Kong et al., 2012). This theory was carried through to the early 2000s (Crow, 2000) when the development of whole genome sequencing (WGS) allowed true DNM rates to be observed.

Using the constant division model, the estimated DNM for 20-year-olds and 60-year-old males would be 40 and 91, respectively (Goldmann et al., 2018).

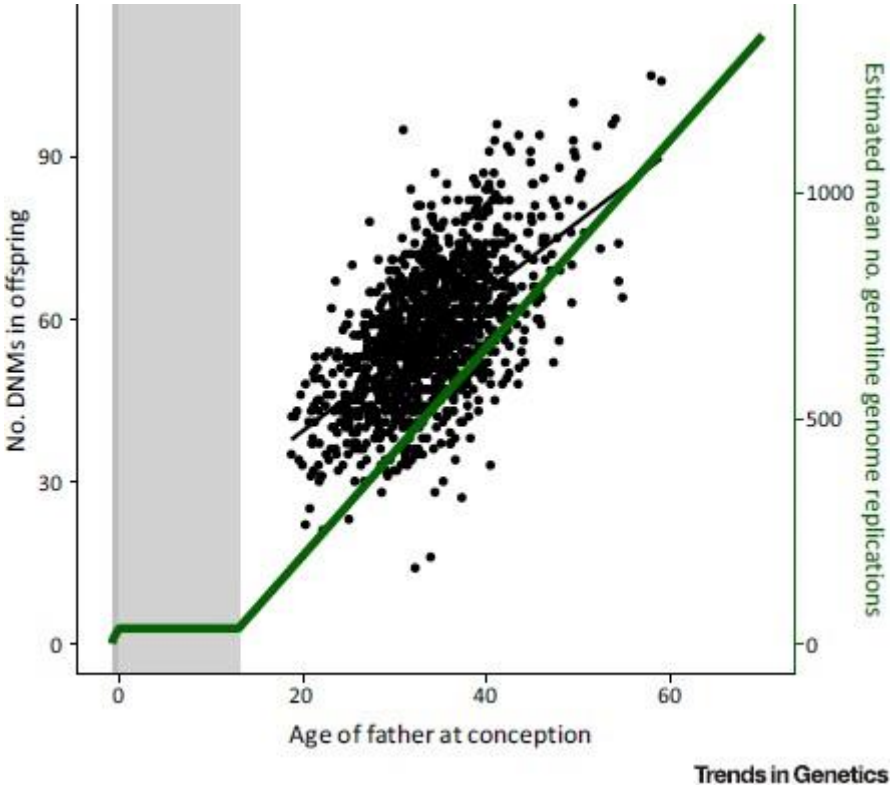


Figure 2: Estimated genome replications in the male germline versus number of de novo mutations (DNM) in offspring (Goldmann et al., 2019). Permission for use granted by co-author C. Gilissen

However, Figure 2 shows this estimation of de novo mutations (green line) based on constantly dividing sperm cells vs the actual observed number of de novo germline mutations (the average of these is shown as a black line). While this disparity between actual observed and estimation of germline DNM has been seen before, Goldmann notes that the original number of estimated cell divisions in Heller & Clermont 1963, used in early DNM research, is outdated. Through the development and use of modern genetic and epigenetic technology, researchers now have a better understanding of the sperm division process. The current model estimates 44 and 71 as the number of DNM passed offspring from 20 and 60-year-olds (Sharma et al., 2019). Having more accurate estimates and an understanding of underlying processes is key to building and using more accurate DNM identification software and pipelines.

Kong et al., used Genome Analysis Toolkit (GATK) and R statistical modelling to determine that the father's age at conception had the greatest correlation with the number of DNMs observed in offspring. Goldmann et al., 2019 agreed with this correlation but with more developed bioinformatic pipelines and tools to restudy rates of spermatogenesis, Goldmann et

al., 2019 achieved a more accurate number of replication estimates. This led to an estimated de novo rate that was much closer to the actual rate observed in the study population.

Goldmann et al., 2019 defined different types of DNMs, characterised by a range of factors, that modern algorithms are starting to be able to distinguish between. Some of these de novo classifications included embryonic, post-zygotic, germline and somatic. Somatic DNM are mutations that occur post-zygotically to post-natally and throughout life (Mohiuddin et al., 2022) in autosomal cells (non-sex cells). These mutations can be important to the life and well-being of the individual, with some somatic mutations being associated with cancer, (do Valle et al., 2016; Goldmann et al., 2018), however, because they occurred after the formation of the germline, they cannot be passed on to any offspring.
Germline mutations are the mutations that occur in the egg or sperm of an individual's parents and are present in most (approx. 50% or higher) of the individual's cells without being present in their parent (Goldmann et al., 2019; Ségurel et al., 2014). These types of DNM can be passed on to offspring and are how a large amount of new genetic material is introduced into a species.

Each type of DNM presents a unique mutational signature and characteristics. However, due to similarities in genetic makeups distinguishing between these types of DNM is still a struggle for our current bioinformatics technology.
Goldmann et al., 2019 concluded that a more specific understanding is "*not only important for our fundamental understanding of biology and human evolution but also have implications for human health and disease*" (Goldmann et al., 2019, p.836). Expanding on the uses of more specific DNM research, being able to track these mutations can also help with breeding programs for animals i.e., cows (Harland et al., 2017).
In addition to typical DNM, there are cases where an individual can have a DNM without fully expressing the phenotype, this is called mosaicism, seen in Figure 3 section C. Mosaicism occurs when an individual has genetically different cells within a single tissue. For mosaic DNM this means only a subset of an individual's cells (including germline) contain a DNM, typically occurring during embryo development but before germline formation. For example, this could result in the DNM occurring in 10% of the cells of an individual, but the individual's children, who inherit the DNM as a normal heterozygous variant, will express the mutation in all cells. This leads to a bigger impact on the child

compared to the parent who originally had the DNM (Mohiuddin et al., 2022; Veltman & Brunner, 2012; Yochem & Herman, 2005).

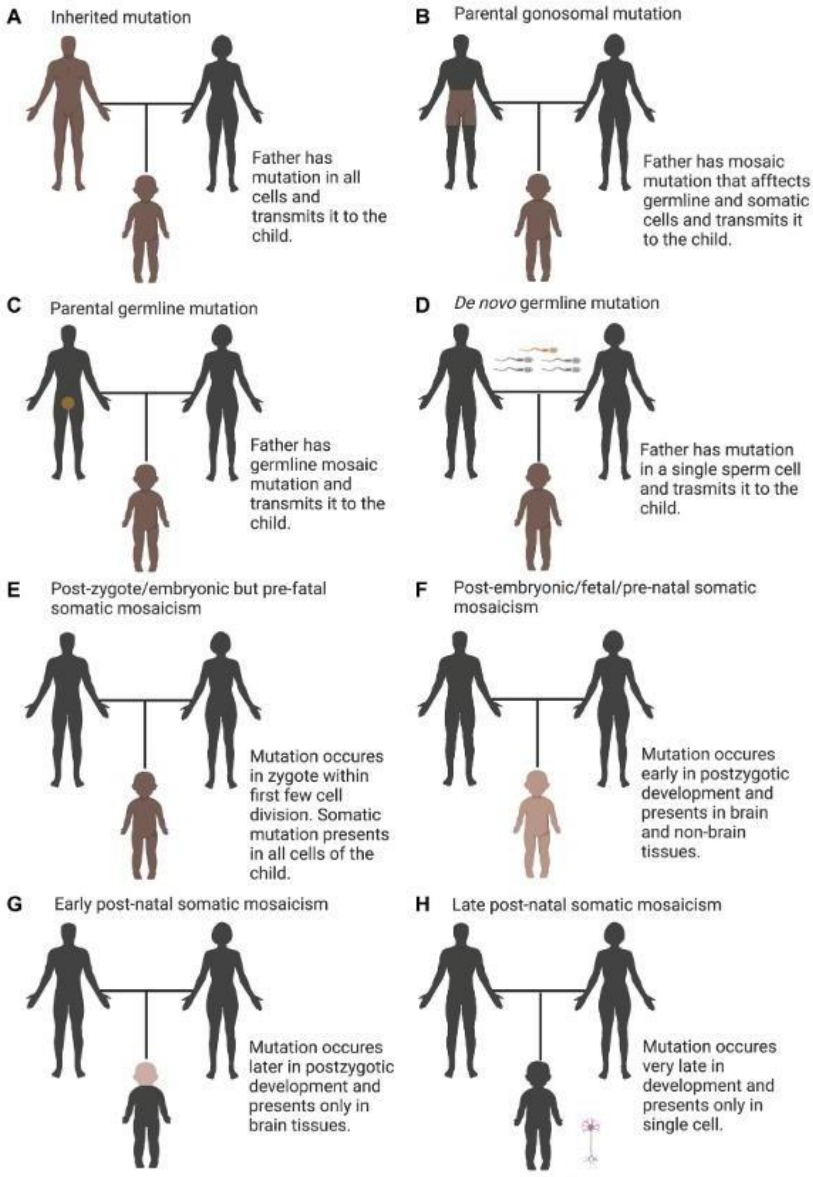


Figure 3: Types of germline, somatic and mosaic de novo mutations and their heritability (Mohiuddin et al., 2022).

The combination of germline, mosaic, and somatic mutations present in an individual's DNA complicates the analysis of germline DNMs and the estimation of the true generational mutation rate that is key to our understanding of a species and its evolution. This is where the continued development of bioinformatics software becomes crucial.

Bioinformatics

The field of bioinformatics began more than 60 years ago with the foundations for the field coming in the early 1960s, before DNA could even be sequenced (Gauthier et al., 2019).

Some of these foundations include computational methods for protein sequence analysis which later lead to the analysis of DNA after the advancement of sequencing and computer science seen in the 1990s-2000s (Gauthier et al., 2019).
Today, bioinformatics combines computer science and biology, using computational power to analyse and manage many different forms of biological data. A significant area of modern bioinformatics is the analysis of Whole Genome Sequences and other nucleotide data produced by NGS technologies. By using computational methods researchers can process, manage, and analyse the large amounts of data produced by NGS. This could be for read mapping (matching sequence reads to a reference genome), variant calling (taking note of differing alleles at specific locations) or other related processing commands.
The role of bioinformatics has grown due to the availability of reference genomes over the Internet, coupled with extensive, accurate and continued reduction in cost of NGS. This has led to an exponential influx of data with a rapid proliferation of bioinformatics tools (Gauthier et al., 2019; Gill et al., 2016). Today, two common software packages to process large-scale raw sequencing data by industry standards are HTSlib-based Samtools and BCFTools packages (Danecek et al., 2021) and GATK (De Summa et al., 2017; McKenna et al., 2010; *Releases · Broadinstitute/Gatk*, n.d.).

Samtools is one of the older bioinformatic NGS pipelines, developed in 2009. It was originally developed by the 1000 genome project to create new data formats for compact storage and efficient analysis of sequence data files used to develop a catalogue of common human genetic variation (Danecek et al., 2021; Li et al., 2009; *Samtools*, 2012/2022). Commonly known for its three file formats, SAM, BAM, and CRAM that support single and paired-end reads and scale up to alignment sets of 10^{11} base pairs (Li et al., 2009). To do this the reads are mapped and stored against a reference genome (Trapnell & Salzberg, 2009). A reference genome sometimes called a reference assembly, is a high-quality whole genome file made up of a single or multiple individuals' DNA. This genome is an idealized representation of an organism that can be used to align sequencing reads to minimise the computing power required compared to spontaneous overlap read assembly (Ekblom & Wolf, 2014; Rice & Green, 2019).
Samtool's original file type SAM is made up of header and alignment sections. The alignment section has 11 compulsory fields of information (as seen in Table 1) and several optional fields and tags. (Li et al., 2009).

Table 1:Mandatory fields in the SAM format (Li et al., 2009)

No.	Name	Description
1	QNAME	Query NAME of the read or the readpair
2	FLAG	Bitwise FLAG (pairing, strand, mate strand, etc)
3	RNAME	Reference sequence NAME
4	POS	1-Based leftmost POSition of clipped alignment
5	MAPQ	MAPping Quality (Phred-scaled)
6	CIGAR	Extended CIGAR string (operations: MIDNSHP)
7	MRNM	Mate ReferenceNaMe ('=' if same as RNAME)
8	MPOS	1-Based leftmost Mate POSition
9	ISIZE	Inferred Insert SIZE
10	SEQ	Query SEQuence on the same strand as the reference
11	QUAL	Query QUALity (ASCII-33=Phred base quality)

Samtools also created the smaller file format BAM. This is the binary counterpart to the SAM format, keeping all the information in a compressed BGZF library (Genome Research Limited, 2022; Li et al., 2009). From the 1.0 version update, CRAM was also introduced as an even smaller compressed file size. This update also saw all the commands work automatically with all 3 file types for processing without conversion (Danecek et al., 2021). The real power of the Samtools software package is the ability to parse, manipulate, and efficiently query these large data files.

In conjunction with Samtools, Bcftools is a counterpart software package that converts the aligned BAM/CRAM files into summarised variant-only files. The original purpose of this was to separate the computer-intensive tasks of variant calling into separate steps (Danecek et al., 2021). This information would then be saved in variant calling format files, VCF, or a more compressed binary version BCF. Since its inception, Bcftools has now become a full package with multiple commands and plugins to manipulate variant files.

Another popular bioinformatic software is Genome Analysis Toolkit (GATK) which was developed by Broad Institute (McKenna et al., 2010). Once SAM files became the standard for NGS data storage, there was space to simplify upfront coding costs (McKenna et al.,

2010). Taking advantage of this new uniformity, GATK was able to build, and continue to update, an engine of pipelines that anyone can use and adjust with relative ease.
GATK is based on the thinking of MapReduce (De Summa et al., 2017; McKenna et al., 2010).
MapReduce is a programming model used for processing and generating large datasets by breaking the data into smaller subsets that can be processed simultaneously for increased processing performance and speed (Dean & Ghemawat, 2008; Elmeleegy et al., n.d.). Using this, GATK uses parallelisation, merged file input, and other techniques to reduce computational power, time, and error rates for large-scale projects. Parallelisation is a computing technique where a large problem can be broken into smaller problems and run simultaneously for results in a faster time (Asanovic et al., 2006). The merged file input is a GATK process that allows multiple BAM files to be merged or other sequencing output to be clustered together for a single analysis without having to alter the input data beforehand. This can be very useful when trying to use or manage a composite of data from a variety of sources.
Other smaller bioinformatic packages such as Rufus, DenovoCNN, and Denovogear have been implemented in the DNM area of research and identification and are discussed in further depth below.

De Novo Mutation Identification Software Programs

There are many different new software packages available for DNM identification. For example, RUFUS (J. A. R. Farrell, 2014/2022; Ostrander et al., 2018), a k-mer-based approach, DenovoCNN (*DeNovoCNN*, 2021/2022; Khazeeva et al., 2022), a machine learning neural network approach, DenovoPedFilter (aeonsim, 2014/2019; Harland et al., 2017), a multigenerational Bayesian statistics method and DenovoGear (*DeNovoGear – Estimating de Novo Mutations from Related Individuals and Cells*, 2013/2022; Ramu et al., 2013), a single generation trio (mother, father, offspring) Bayesian method. These software programs were originally designed for a single project/purpose and while parameters can be adjusted, they are not professionally monitored, updated, or supported. This means that once they are published with the corresponding paper, they are open-access and are free to use but no further development is expected. Unlike many other modern professional software packages, these research quality programs do not undergo the same level of quality assurance. Particularly for DNM research which does not have a foundation of older software

to compare to or build on. As this is still a rapidly developing area of research, no standard methods have been agreed on and widely validated and accepted.

DeNovoGear

One of these new methods and programs is called DeNovoGear, a statistical-based software (*DeNovoGear – Estimating de Novo Mutations from Related Individuals and Cells*, 2013/2022; Ramu et al., 2013). It uses Bayesian statistical modelling to predict the probability (p) that a variant in a trio's offspring (mother, father and offspring/child) is a DMN. Unlike other statistical modelling types, Bayesian statistics do not distinguish any of the unknown variables in the model (Wilkinson, 2007). This makes Bayesian statistics uniquely able to handle biology's complex, non-linear data problems and observations. In Bayesian statistics, all parameters, variables, and observations are treated consistently for the outcome. Historically Bayes' theorem (**Equation 1**) was limited greatly by computational power.

$$p(B|A) = \frac{p(A|B)p(B)}{p(A)}$$

Equation 1:Bayes' Theorem (Lynch, 2007). A, B = events. P = probability

With the vast modern improvements and use of high-power computing, Bayesian statistics' ability to update prior models and beliefs with new data, has become a staple in mainstream scientific modelling.

According to Ramu et al., 2013, the "basic" approach to DNM identification is to identify the mutation site/type, for the trio of samples and compare for differences between the parents and the offspring. DenovoGear builds on this so-called "basic" approach by adding to their model, individual genotype likelihoods, transmission probabilities and probability of observing a SNP, INDEL or DNM at any specific site on the genome (Ramu et al., 2013), see Equation 2. These parameters can be altered to either increase or decrease calling sensitivity i.e., the accuracy of identifying DNM as true, false positive or not identifying them.

This current framework is limited to trios and matched simple paired samples.

$$L(D) \; \alpha \; P(D_M|G_M)P(D_F|G_F)P(D_C|G_C) \;\; x \;\; P(G_C|G_M, G_F)P(G_M, G_F, R)$$

Equation 2: DenovoGear DNM identification probability model (Ramu et al., 2013). L = likelihood. Subscripts indicate genotype (G) from mother (M), father (F) and child I. R = public reference genome sequence.

For DenovoGear's algorithm, the G stands for genotype (subscript M for male father, F for female mother and C for child). The end of Equation 2, P(GM, GF, R) is from drawing two

sequences and comparing them to the base present in the reference sequence I, this value can also be adjusted by users entering a known mutation rate for the population. P(GC | GM, GF) is the transmission likelihood from parent to offspring and finally another prior can be found in the assumed DNM probability represented by P(GC | GM, GF).

Table 2: Comparison between different de novo mutation callers using whole genome sequence (Ramu et al., 2013)

Caller	Posterior Probability cutoff	Known Germline DNMs	Known Somatic DNMs	Known False Positives	Total DNM Calls	Sensitivity (Germline)	Sensitivity (Somatic)	False Discovery Rate
Denovogear	0.5	49	935	259	5732	1.0000	0.9821	0.8283
	0.6	49	931	257	5438	1.0000	0.9779	0.8198
	0.7	48	931	256	5190	0.9796	0.9779	0.8114
	0.8	48	930	252	4851	0.9796	0.9769	0.7984
	0.9	48	930	250	4474	0.9796	0.9769	0.7814
	1.0	47	929	247	4140	0.9592	0.9758	0.7643
Polymutt	0.5	48	878	263	6215	0.9796	0.9223	0.851
	0.6	48	878	263	6215	0.9796	0.9223	0.851
	0.7	48	878	263	6215	0.9796	0.9223	0.851
	0.8	48	878	263	6215	0.9796	0.9223	0.851
	0.9	48	878	263	6215	0.9796	0.9223	0.851
	1.0	48	878	263	5938	0.9796	0.9223	0.8441
GATK	0.5	47	932	276	18421	0.9592	0.979	0.9469
	0.6	47	932	276	18421	0.9592	0.979	0.9469

Caller	Posterior Probability cutoff	Known Germline DNMs	Known Somatic DNMs	Known False Positives	Total DNM Calls	Sensitivity (Germline)	Sensitivity (Somatic)	False Discovery Rate
GATK	0.7	46	929	273	17036	0.9388	0.9758	0.9428
	0.8	46	928	273	16410	0.9388	0.9748	0.9406
	0.9	46	921	270	15141	0.9388	0.9674	0.9361
	1.0	46	913	265	13440	0.9388	0.959	0.9286
Samtools	0.5	49	890	328	235134	1.0000	0.9349	0.996
	0.6	49	890	326	20176	1.0000	0.9349	0.9957
	0.7	49	890	325	207307	1.0000	0.9349	0.9955
	0.8	49	890	321	167913	1.0000	0.9349	0.9944
	0.9	49	890	313	111142	1.0000	0.9349	0.9916
	1	49	890	292	51246	1.0000	0.9349	0.9817
NaiveCaller	NA	49	936	320	144424	1.0000	0.9832	0.9932

DenovoGear's algorithm was compared to 4 other DNM-identifying software pipelines when using WGS (Table 2). DenovoGear at lower posterior probability cut-off (0.5 and 0.6) outperformed the other callers for Germline sensitivity. With somatic DNM sensitivity, DenovoGear 0.5 posterior cut-off was second only to the NaiveCaller.

After testing and comparison, DeNovoGear was published on GitHub for further testing and development. Instructions for installation, running and expected output can be found on the readme file of the repository *DeNovoGear – Estimating de Novo Mutations from Related Individuals and Cells*, 2013/2022. The implementation of this algorithm has also been developed as a plugin for the BCFtools software package (Danecek, 2020) in association with SAMtools.

PedFilter

PedFilter is a software designed by Chad Harland in 2016 for the Damona project, a large project which studied the rate of germline DNM in cattle (aeonsim, 2014/2019; Harland, 2018; Harland et al., 2017). Originally designed for analysing cattle PedFilter is made up of a series of scripts designed to use GATK-produced VCF files across a 4-generation pedigree and provides DNMP algorithm (Venn et al) as a means of ranking candidate variants when multiple-generation pedigrees are not available. The scripts generate linkage between grandparents, parents, proband and grand offspring, as seen in Figure 4.

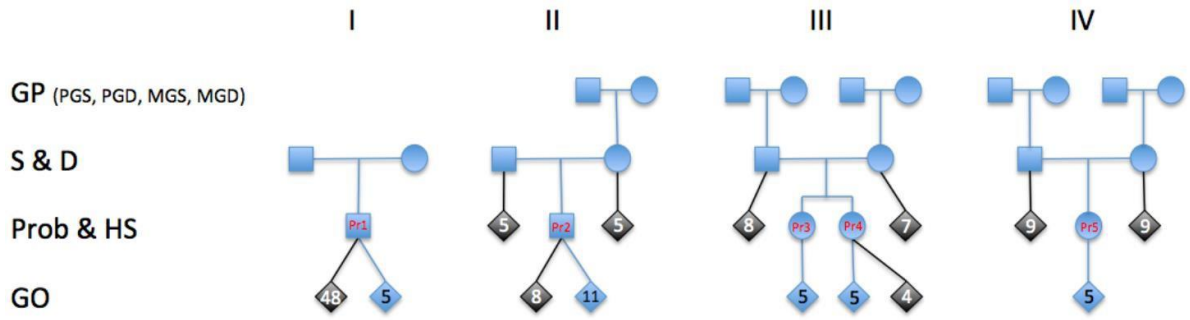


Figure 4: Four pedigrees (I, II, III, IV) used for the detection of dnm's. GP: grand-parents (PGS: paternal grand-sires, PGD, 325 paternal grand-dams, MGS: maternal grand-sires, MGD: maternal grand-dams), S: sires, D: dams, Pr: probands, HS: half-sibs (of the 326 proband), GO: grand-offspring. The five probands are labelled in red. Animals in blue were genome-sequenced at average depth of 23 327 and used for the detection of dnm's. Animals in gray were used for confirmation by whole genome (average sequence depth of 20) or 328 targeted sequencing (see Supplemental Methods). DNA was extracted from venous blood for females, and semen from males, except for 329 Proband 1 for which both semen and blood DNA were analysed (Harland et al., 2017).

PedFilter uses a heuristic approach to search for variants with the following properties, high-quality score (>=100), genotypes of 0/1 (alternate from a reference sequence), the alternate allele not found in the grandparents or parents and reads with the alternate allele found in the proband's offspring (Harland et al., 2017). The program also uses population data to determine the genotype frequency at that position across unrelated individuals (Harland, 2018).

The pipeline was tested against 11,255 (10,093 being SNPs) simulated DNMs generated from variants identified by the 1000 Bulls project (Hayes & Daetwyler, 2019). PedFilter found 9,325 DNM candidate variants for an 83% sensitivity (8,409 SNPs for 81% sensitivity) (Harland, 2018; Harland et al., 2017).

PedFilter also has a unique perspective on DNM compared with the other software listed. Unlike DenovoGear and DenovoCNN, which were designed using human genetics and for

humans, PedFilter was designed for bovines using bovine genetics. This allowed Harland et al., 2017 to account for the role of mosaicism and how it can present and affect population genetics when dealing with such large breeding scales as seen in the New Zealand dairy industry. This is where the use of the 4^{th} generation (grand offspring) sets PedFilter apart from other DNM pipelines. For a DNM to be considered a true DNM and not inherited from a mosaic parent, the DNM dosage (proportion of reads carrying the mutant allele at the DNM site) for the proband, should be less than 50%, with grand offspring who inherited the gene to show as equal to 50%. When PedFilter was used on the pedigrees found in Figure 4, the 31 possible DNMs found had a mean dosage of 0.26 in the proband and 0.52 in the grand offspring. This was found to be a significant difference with a p-value of less than 10^6 (Harland et al., 2017).

DenovoCNN

DenovoCNN is a machine-learning-based software that uses computational neural networks to learn to recognise potential DNM (Khazeeva et al., 2022). Machine learning (ML) is the branch of computational algorithms designed to mimic human intelligence. It is an interdisciplinary field of research focusing on developing and using algorithms that allow computers to learn and can refer to any algorithm that allows a computer to improve a task the more times it is used/run on data (El Naqa & Murphy, 2015; Larrañaga et al., 2006; Zhang & Rajapakse, 2009).

For bioinformatics, this could mean using different kinds of biological data to train computers to find and identify different information. In DenovoCNN's case, it takes variant genetic data and is trained to find differences between the proband and its parents to identify potential DNM.

The training stage of the ML algorithm is generally running a basic idea of the algorithm on small samples of known data. As the number of times the algorithm is run grows and the algorithm's output gets closer to the expected output, more data can be added to help prepare the computer to recognise new or different scenarios. In recent years with the expansion of NGS and the large amount of data being produced the idea and practice of ML for bioinformatics has grown as well. Once trained these bioinformatic algorithms can be hugely useful in identifying required information or can be retrained relatively easily to keep up with ever-changing research (Chen, 2019; Kavikondala et al., 2019).

For DenovoCNN, the algorithm was developed as a convolution neural network that encoded the alignment of sequence reads for a trio as images (Khazeeva et al., 2022). Khazeeva et al., 2022 paper identified methods such as DenovoGear and TrioDeNovo for modelling the probability of transfers using mutation rate priors. While these methods can yield high sensitivity, they lose out on specificity because of noise in NGS data. This in turn lets in more false positives and typically requires subsequent checks and filtering. Khazeeva et al., 2022 claim that deep learning has grown in popularity because the approach of converting genomic data into a representative image has improved on many previous genomics applications for the same jobs. With all these benefits, ML also has some setbacks. Substantial set-up is required to start building and training new models, with a lot of manual checks required to do this.

Figure 5 shows four main steps to the model-building of DenovoCNN: training data, image encoding, network training and validation processes. For the training data, a large population is required to start (e.g. 6067 trios for this model). Initial image encoding identifies possible DNM (e.g., 13,068 DNM were identified). This data then was visually assessed using Integrative Genomic Viewer (IGV) (Robinson et al., 2011).

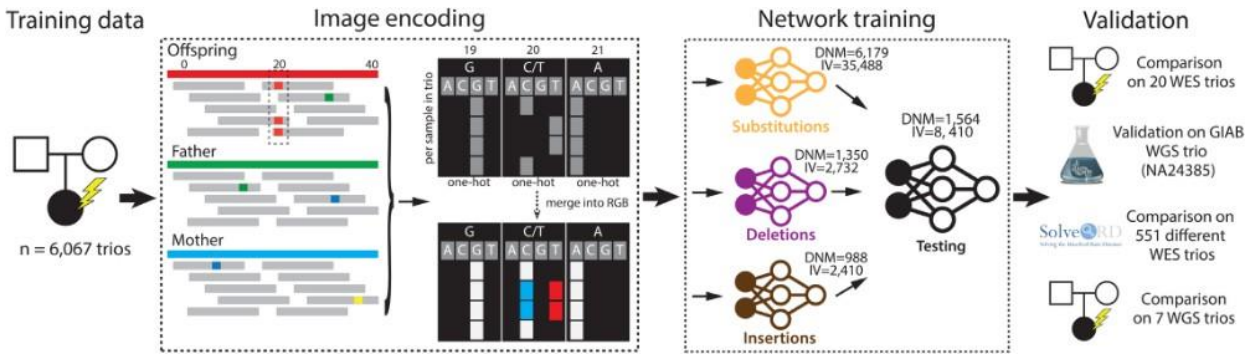


Figure 5:Overview of the four method steps for DeNovoCNN model building (Khazeeva et al., 2022). 'Image encoding' - each colour channel corresponds to a variant at a specific genomic location from the child (red), father (green), and mother (blue) data. Every row encodes a separate sequencing read.

These visual assessments helped identify false positives and confirm true positives. This knowledge of true and false positives was added and used to train the neural network. Each iteration improves at recognising the signs of false positives and identifying true positives that other software may cut off. This training was performed on the previously known data so accuracy could be measured. After training, the new model was tested and validated by running DenovoCNN on new unknown trios. These new trios were also run with DenovoGear (Ramu et al., 2013), DeepTrio (Kolesnikov et al., 2021), GATK

PossibleDeNovo (DePristo et al., 2011), and an in-house software based on Samtools. The results of this validation testing can be seen in Table 3.

Table 3: Results of the comparison on the GIAB dataset (Khazeeva et al., 2022)

Tool	Calls	TP	FP	TN	FN	Sensitivity/ Recall	Specificity	Precision	Accuracy
DeNovoCNN	1233	1198	35	640	125	90.55	94.81	97.16	91.99
DeNovoGear-0.5	1346	1063	283	392	260	80.35	58.07	78.97	72.82
DeNovoGear-0.9	1161	1047	114	561	276	79.14	83.11	90.18	80.48
DeepTrio (DeepVariantWGS, BQSR)	210	176	34	641	1147	13.3	94.96	83.81	40.89
DeepTrio (DeepVariantWGS, no BQSR)	268	240	28	647	1083	18.14	95.85	89.55	44.39
DeepTrio (DeepVariant unfiltered, BQSR)	1207	1127	93	582	194	85.19	88.15	93.27	86.19
DeepTrio (DeepVariant unfiltered, no BQSR)	1222	1129	93	582	194	85.34	86.22	92.39	85.64
GATK	1338	1171	167	508	152	88.51	75.26	87.52	84.03
GATK HC	1257	1149	108	567	174	86.85	84.0	91.41	85.89
In-House	1293	1195	98	5777	128	90.33	85.48	92.42	88.69

The Problem/Goal

De Novo Mutation research is a budding area of genetics and informatics made possible by the rapid advancements in sequencing technologies and their reducing cost. The last decade has seen the development of DNM research starting with Conrad et al., 2011 and Kong et al., 2012 and then hitting another boost in 2017, with eight new papers published and a few every year after that (Bergeron et al., 2022). Of all the published papers so far, a heavy majority focus on humans (11/21) with most other species studied, having only one paper. Across most of these papers, there are shared problems with their DNM research. This is finding the balance between high sensitivity, to find all true DNM, and high precision, to avoid false

positives. This has led many researchers to design their own software, using different methods with differing emphasis on sensitivity to precision.

For Bergeron et al., 2022 a comparative study was undertaken where the same data (taken from Bergeron et al., 2021, Rhesus Macaque study) was sent to five research institutes, who designed their own methodologies to identify candidate DNM and estimate a germline mutation rate. Dubbed the "Mutationathon" (Bergeron et al., 2022), the aim was to examine the estimated mutation rates produced by the institutes and highlight how the systemic differences in programs impacted the outcome of data. Every team followed the same five steps in building their pipeline: sampling, read alignment, variant calling and genotype likelihoods, DNM detection, and Per generation mutation rate estimation.

Even following the same basic steps, the differences in filtering, see Table 4, lead to very different results from each group for the same dataset.

Table 4: Site-specific and sample-specific filters used by the different groups to detect de novo mutations (DNMs) (Bergeron et al., 2022)

Research Group	Candidate DNMs	Site Specific Filters	Sample specific filters	Additional filters
CV	18	GATK Best Practices hard filter criteria	0.5 × dpind < DP < 2 × dpind GQ > 40 AD > 0 0.25 < AB < 0.75	
RW	22	QD < 2.0 MQ < 40.0 FS > 60.0 SOR > 3.0 MQRankSum < –12.5 ReadPosRankSum < –8.0	20 < DP < 80 GQ > 20 AD > 0 0.35 < AB	Alternative allele on both strands
TT	27	Remove variants in recent repeats or in homopolymers of AAAAAAAAAA or TTTTTTTTTT	DP > 10 GQ > 20 AD > 0 0.25 < AB	Overlap three different variant callers Filter on LCR
LB	28	QD < 2.0 FS > 20.0 MQ < 40.0 MQRankSum < –2.0 MQRankSum > 4.0 ReadPosRankSum < –3.0 ReadPosRankSum > 3.0 SOR > 3.0	0.5 × dpind < DP < 2 × dpind GQ > 60 AD none 0.3 < AB < 0.7	Manual curation (six candidates removed)
SB	32	FS > 30.0 MQRankSum < –10 MQRankSum > 10 ReadPosRankSum < –2.5 ReadPosRankSum > 2.5 BaseQRankSum < –13 BaseQRankSum > 13	10 < DP < 2× dpind GQ > 55 AD > 0 0.3 < AB	Alternative allele in both strands. lowQ AD2 > 1

LB: Lucie Bergeron; SB: Søren Besenbacher; CV: Cyril Versoza; TT: Tychele Turner; RW: Richard Wang

Bergeron et al., 2022 noted that no single institute identified all the DNM without false positives. Each group set their sensitivity to precision ratio differently and it showed. It was iterated that due to the lack of standardisation, there will always be an impact on how germline mutation rates are estimated as well as the accuracy of finding candidate mutations. This sentiment has been expressed across several other DNM papers (Acuna-Hidalgo et al., 2016; Goldmann et al., 2019; Harland, 2018; Harland et al., 2017; Khazeeva et al., 2022; Ostrander et al., 2018; Ramu et al., 2013; Ségurel et al., 2014; Veltman & Brunner, 2012). Bergeron et al., also noted when comparing mutation rates between papers, even within the same species, there are several aspects to consider. They go to list a few best practice guidelines seen in Table 5.

Table 5: Information that should ideally be reported when presenting results of DNMs (Bergeron et al., 2022)

Step of the Analysis	**Information to report**
Sampling and sequencing	Type of Sample (tissue etc.)
	Storage Duration, buffer, temperature
	Type of Library preparation
	Average Sequencing Cover
	Sequencing technology and read lengths
Alignment and post-alignment processing	Trimming of adaptors and low-quality reads
	Reference assembly version Autosomes only or whole genome
	Mapping software and version
	Duplicate removal software and version
	Base quality score and recalibration (Yes/No)
	If yes, which type of data used as know variants
	Realignments around indels?
	Other filters
Variant calling	Software and version
	Mode: joint genotyping, Gvcf blocks? Gvcf in base-pair resolution
Detecting De Novo Mutations Detecting De Novo Mutations	Site filters on vcf files and justification
	Individual filters, threshold, and remaining candidates after each filter
	False discovery rate esitmates method: PCR validation? Manual Curation? Transmission rate deviation? Removal f LCRs, cluster mutations or recurrent mutations?
Mutation rate estimation	Callable genome estimation method: Files used? Filters taken into account?
	False-Negative rate estimation method: simulation? Filters? Probability?

The “Mutationathon” (Bergeron et al., 2022) confirmed and provided some insight into the impact methodologies have. It also shed light on how difficult comparing data if the information in Table 5 is not provided.

One challenge with DNM research that Bergeron et al., 2022 did not address is that developing new pipelines is very time-consuming and comparison and further development between different pipelines, is difficult even with guidelines.

The aim of this thesis is to use a single set of bovine trios (provided by LIC) to test four existing pipelines against each other. In doing this find a preferable balance of sensitivity and precision for the open-source programs. Thereby providing an option with comparative data about existing software for use.

Chapter 2: Methods

Setting up an approved NeSI Account

The New Zealand eScience Infrastructure (NeSI) provides a shared high-performance network computer base in Wellington, New Zealand with collaborators from the University of Auckland, NIWA, the University of Otago, Manaaki Whenua and the Ministry of Innovation and Employment (*New Zealand EScience Infrastructure*, 2019.). Access to this database was gained by applying to the NeSI for a My NeSI account (*Creating a NeSI Account Profile*, 2022). Using a Tuakiri institute login, set-up questions were asked about contact details and current role in the institute (*Tuakiri - Trust and Identity :: REANNZ*, 2022). Next, access to an existing project was applied for. To do this the project name and number were added to the application. This information is sent to the manager of the applied-for project along with the account information for approval and access. Once approval was met, access was granted, and setup was complete. There is an alternate application process for applying for space to start a project, but this was not needed or completed for this study.

Selection of Genetic data (Cram Files)

All genetic data used in this study were selected from Livestock Improvement existing data. All data had been collected humanely for animal health and breeding reasons. Thus, animal ethics approval from the University of Waikato was not required. This study used pre-existing genetic files. Chad Harland picked the bovine trios randomly from Whole Genome Sequencing files that met a read depth of 20 or more and had complete trios (mother, father, offspring). A total of 6 trios were studied.

Software Selection

Originally four software packages were selected for comparative analysis of the 6 trios, shown in Table 6.

Table 6: Originally selected software packages and links. Ordered by when they successfully were run (or dropped)

Name of Package	Software location/Website	Research paper Name and Link
DenovoGear	https://github.com/ultimatesource/denovogear	Ramu, A., Noordam, M. J., Schwartz, R. S., Wuster, A., Hurles, M. E., Cartwright, R. A., & Conrad, D. F. (2013). DeNovoGear: De novo indel and point mutation discovery and phasing. *Nature Methods*, *10*(10), Article 10. https://doi.org/10.1038/nmeth.2611
PedFilter	https://github.com/aeonsim/denovoPedFilter	Harland, C. (2018). *Germline mutations in Bos taurus* [ULiège - Université de Liège]. https://orbi.uliege.be/handle/2268/223689
DenovoCNN	https://github.com/Genome-Bioinformatics-RadboudUMC/DeNovoCNN	Khazeeva, G., Sablauskas, K., van der Sanden, B., Steyaert, W., Kwint, M., Rots, D., Hinne, M., van Gerven, M., Yntema, H., Vissers, L., & Gilissen, C. (2022). DeNovoCNN: A deep learning approach to de novo variant calling in next generation sequencing data. *Nucleic Acids Research*, gkac511. https://doi.org/10.1093/nar/gkac511
RUFUS	https://github.com/jandrewrfarrell/RUFUS	Farrell, A. R. (2014). *Expanding the horizons of next generation sequencing with RUFUS* [Boston College]. http://dlib.bc.edu/islandora/object/bc-ir:104176

Each one operated using different methods of calculating or identifying De Novo Mutations (DNM). After preliminary analyses, Rufus and DenovoCNN were omitted, and the focus turned to analysing output data from Pedfilter and DenovoGear exclusively.

DenovoGear

The DenovoGear library/repo was downloaded onto NeSI from GitHub (Ramu et al., 2013). Following the ReadME file from the repo, installation to NeSI was attempted. The first round of testing and code was run on a single Trio for testing purposes. This was to save time and computing power for when errors arose, or unformatted output was received.

Once the running pipeline was set up, it was amended for bulk running on all trios. However, this bulk run was never used, as during the testing phase, DenovoGear was updated to be accessible as a plugin for Samtools (Li et al., 2009). LIC used this option and updated all major VCF files within the company to include this tag. Also, by being a plugin, DenovoGear

was easier to access and use on NeSI as it became a program maintained by the NeSI themselves.

After this update, a Samtools Query command (Appendix A.i) was used to select relevant information about the trio samples. This information included chromosome number, genome position, reference allele, alternate alleles, sample ID, Genotype, and DenovoGear score. This initial output from this query was later refined to remove any unnecessary data using Linux command line. Removed data included bash output messages, syntax abnormalities, and errors.

Finalised Output data was uploaded to Jupyter Notebooks for further processing and analysis, see Appendix B.

PedFilter

PedFilter working directory was downloaded from the GitHub repo (aeonsim, 2014/2019) onto NeSI. Following the readMe instructions for set up and running, the average read depth per sample was calculated, see Appendix A.ii. Separating samples into 29 autosomal chromosome files was executed to speed up processing time and lower computing costs. PedFilter also relies on population data so a set population of 500 animals were selected from the whole LIC population data. The smaller population was used again to save processing time and computing power. Using 500 animals provided a large enough sample population for the 6 trios being studied. The population sample was selected based on similar criteria to the trios, requiring a minimum read depth of 20 and being whole genome sequences.

With the list of samples' average depth, population sample and sample data, each sample chromosome was run individually through PedFilter, Appendix A.iii. The output data from the PedFilter was reassembled into one file of all the chromosome data and uploaded to NeSI Jupyter notebooks.

Python Notebook was used to clean and remove unnecessary data, errors, and fix the syntax to match DenovoGear output, see Appendix C.

DenovoCNN

The DenovoCNN container was downloaded into NeSI from the git repository *DeNovoCNN*, 2021/2022. Due to the nature of how DenovoCNN was set up in the repo, NeSI officials were required to set up and install it as a container. Testing using sample data and models provided by the repo was undertaken and was successful. Following this, the research trios underwent GATK Base Recalibrator (*BaseRecalibrator*, n.d.). This process requires an input .bam file, a .fasta sequence reference file and the matching .vcf file to the .bam. This outputs an index table file that is used to further clean the data through Base Quality Score Recalibration (BQSR) (*Base Quality Score Recalibration (BQSR)*, n.d.). This pre-processing step is used to remove systematic errors that can occur via sequencing. This is especially useful when working with data sequenced using different techniques. This was a requirement for running DenovoCNN but is a very time-consuming and computation-heavy task so was not performed on trios for PedFilter or DenovoGear.

After base recalibration, Appendix A.iv and BQSR, Appendix A.v, many attempts to test DenovoCNN using a single research trio and sample models were undertaken and failed. The failures were all linked to issues with accessing and running the container on outer files. Due to constraints set up to protect NeSI, the container was difficult to work with or manipulate to use for the data. Intermittent testing was continued with no success.

It was later decided to shift the work to an LIC base high-performance computer with a large RAM capacity, but subsequent trio files (bam, index and bqsr) overloaded the computer storage.

Before another solution was found and more attempts were made, the original repo was deleted from GitHub and set up under a different account. This caused issues with updates and script links which ultimately led to the decision to forgo further work using this software.

RUFUS

Rufus was the first DNM-identifying software tested for this project. It was designed by J. Andrew R. Farrell as part of their PhD dissertation (Farrell, 2014). Installation, build and run instructions found in the readME file of the GitHub Repository (Farrell, 2014/2022) were followed, noting a disclaimer in the readME file saying the project is under development and

unstable. This proved to be true when RUFUS presented too many issues, most of which were related to the highly specific environments and requirements needed for RUFUS to install, that didn't match the specifications in NeSI. After multiple attempts to bypass these errors or work with them, RUFUS was omitted from further use in the project.

Analysis process

Notebook file uploads, Filtering and Sorting

After running the PedFilter and DenovoGear programs, their respective output files were uploaded to Jupyter notebooks for further processing. However, the DenovoGear file exceeded the JupyterNotebooks maximum upload size and first needed to be refined and some output data removed for faster processing and use. The main filter used to refine the output was removing all output variants with a -inf log-scale probability of being a DNM. This reduced the file size to fit within the 25 MB size limit for Jupyter. Once uploaded, both the PedFilter output and the DenovoGear output underwent further processing and filtering to check for syntax errors so as to provide the most accurate data frame join. See Appendix B for DNG processing, and Appendix C for PedFilter.

The data from both DenovoGear and PedFilter were then merged into a single data frame, using Jupyter. This data frame contained the individual information from both pipelines as well as combining information from DNM found by both programs. Code in Appendix B and C.

Next, using a combination of python data frame (*Intro to Data Structures — Pandas 1.5.2 Documentation*, 2022) extraction and Samtools, trinucleotides (tri-nuc) were generated for DenovoGear data. This was achieved by extracting the chromosome and position of every candidate DNM and adding and subtracting 1 from the position to get a 3-base range using python, Appendix B.

This range was saved in a separate file where Samtools' faidx command was used alongside the reference genome to get the reference genotype tri-nuc, Appendix A.vi. The output from this command was uploaded to NeSI and used alongside the original data frame to exchange the reference allele for the observed alternate allele in the sample. These two tri-nuc frames were formatted and aligned to match the automated tri-nuc data from PedFilter and added back into the data frame. Code found in Appendix A.

Software Identification and Probability Threshold Testing

Once all the data was combined into a single data frame for storage, each candidate DNM was given a tag based on whether they were identified by PedFilter, DenovoGear or both. This tag was used to count the number of candidate DNMs at different probability scores. This was done using Denovo-Posterior for PedFilter and Both data, and DNMsc for DenovoGear and Both. These counts were visually represented to determine if there was a clear score threshold at which the number of possible false positives outweigh the number of potential true positives.

IGV Visual Validation

Validation of candidate DNMs was undertaken using Integrative Genomic Viewer (IGV). All candidates that had been identified by both software packages were visually assessed and noted in an excel file. The top 100 PedFilter candidates were also manually assessed and added to the excel file. Due to the number of same-scoring candidates for DenovoGear, a random selection of top scoring candidates spread over each sample was selected and assessed.

After visual validation, the trinucleotide signature of the assessed candidates was completed. These signature graphs were grouped by validity (yes, no, maybe and (maybe) mosaic) and performed on each software range separately.

Trinucleotide graphing for mutational signature

Using the python data frames, the number of occurrences of each type of trinucleotide was grouped by software tag, counted, and extracted into an excel file. Using excel, forward and reverse trinucleotides were linked, and a mutation ratio was calculated using the genome expected ratio and software observed ratio. The mutation ratio for PedFilter, DenovoGear and Both was then graphed to see the mutational signature noted by each software.

Using the data from IGV validation this process was repeated within each software. The candidates from each software program that was visually assessed using IGV were grouped by their validity and presented as a trinucleotide mutational signature.

Proportion of True, False, Maybe and Mosaic DNMs found by PedFilter, DenovoGear and Both

The validity counts of candidates classified by IGV visual assessments were used to generate proportion Venn diagrams. These diagrams identified the total number of variants of each validity class found across all the assessed candidates and then what proportion of each came from either PedFilter, DenovoGear or both.

Chapter 3: Results

Output data was generated by writing scripts (Appendices A-D) and extracting data from six trios into excel workbooks for manual IGV validation of selected candidate DNMs. A total of 561 candidate DNMs were validated: the top 100 identified by PedFilter ranked by Denovo-Posterior score, 100 candidates identified by PedFilter with a score less than 0.9, 100 randomly selected candidates identified by DenovoGear with a DNMsc of 0.0, 100 randomly selected candidates identified by DenovoGear with a score between -1 and -5 and all candidates identified by Both software programs.

Next, criteria for validating the identified candidate DNMs were established using four validity tags: yes, no, maybe, and mosaic. An example is shown in Table 7 of each. In the following sections, “Candidate DNMs” refers to possible DNMs found by the software that have not been manually validated through IGV.

Table 7: Criteria for assignment of validity tags given to SNP DNM candidates through visual IGV assessment with SNP examples

Validity Tags	Criteria	Chromosome and Position	Example from study
Yes (True positives)	Enough reads (≥ 30%) in the proband contain the Alt allele. No reads containing this allele are found in either parent and no unrelated samples have a greater proportion of reads containing the Alt allele	Chr29:Pos48366041	The sample showed approx. 50%, no reads found in either parent sample or unrelated samples
No (False positives)	Single or multiple reads found in BOTH parent samples, the equal or greater number of reads found in unrelated samples, or insufficient reads found in the proband sample (less than 20%).	Chr19:Pos60507552	Reads found in BOTH parents of the proband and multiple reads found in unrelated samples
Maybe	Reads must not be found in both parents, lower end for the required number of reads (25-40%), some unrelated samples may have read(s)	Chr10:Pos10739198	a limited number of reads (approx. 20%) with a single read in an unrelated sample, but other Alt allele seen in other samples suggest high SNP rates and potential for DNM or a genome assembly error causing false SNPS
(maybe) mosaic	These are DNMs that are or could be mosaics. For this tag, one parent has a low number of reads (1–5) and the number of reads in the proband higher proportion of reads than their parent	Chr17:Pos19980723	A low percentage of reads in the proband, lower number of reads found in one parent, low read numbers in unrelated samples and different Alt allele found in unrelated samples suggesting a higher potential location for DNMs

Software Identification and Probability Threshold Testing

Candidate DNM Found by Both PedFilter and DenovoGear

Candidate DNMs were identified by two different software programs, PedFilter and DenovoGear. The 161 candidates identified by both software programs were grouped and their resulting probability scores were cumulatively counted.

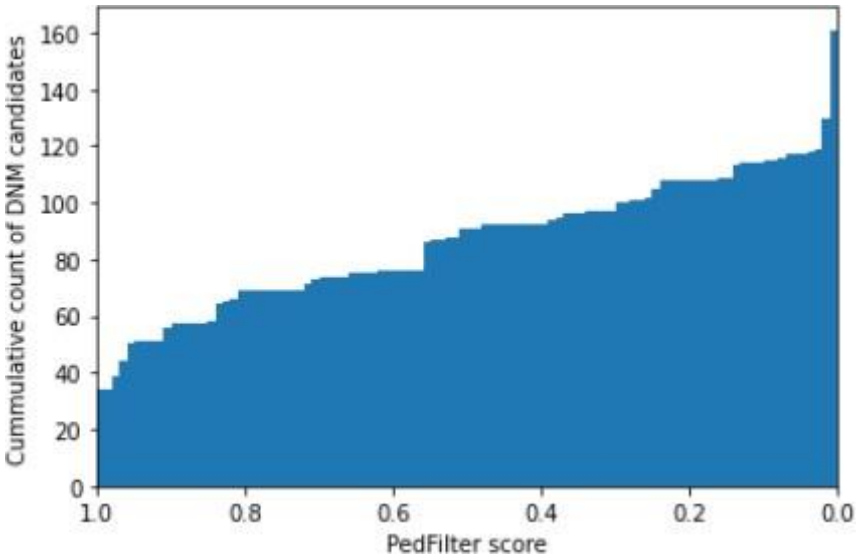


Figure 6: Cumulative count of Pedfilter Denovo-Posterior scores for de novo candidates identified by both software. Each bar is a representation of the number of candidate de novo mutations found at and above that probability score (ie 0.8 range is all candidate DNM with a score of 0.8 or higher). Generated by code in Appendix C.

Figure 6 shows the cumulative count for PedFilter Denovo-Posterior scores. PedFilter scores are based on Bayesian probability and Venn et al., 2014 germline estimation models. Scoring for this has a score of 1 being 100% sure the candidate is a true DNM and 0 being least confident that the candidate is a true DNM.

This figure shows for candidates found by Both software, 21% of candidates were categorised with the most confidence as they had the highest Denovo-Posterior scores (≥ 0.98). The number of DNM candidates increased as the PedFilter score decreased. Thus, the consistent rise of DNMs counted at each Denovo-Posterior threshold means there is no clear cut-off where the number of true positive DNM gained, outweighs the number of false positives being included.

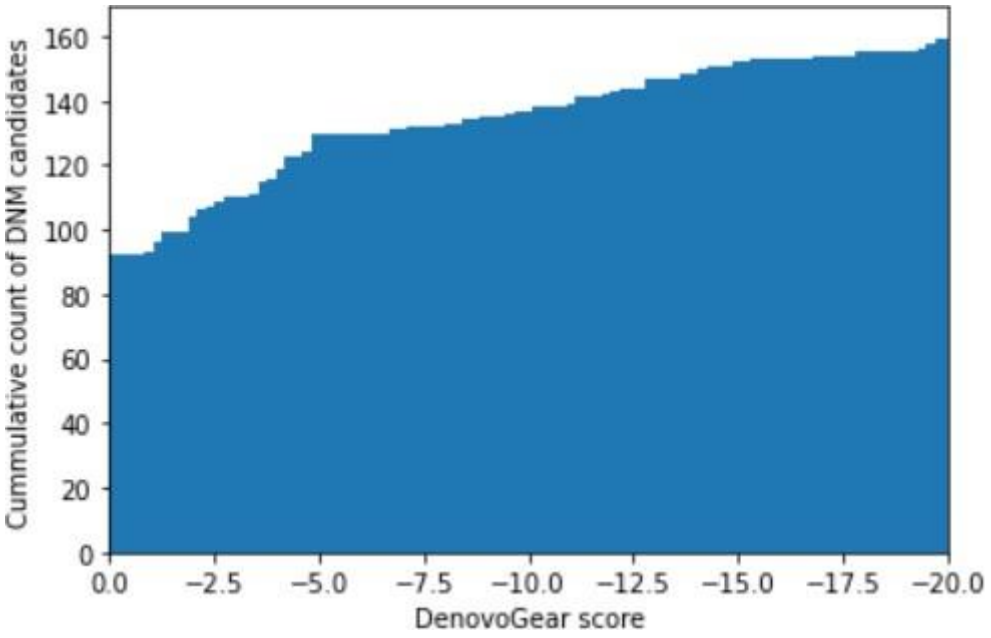


Figure 7: Cumulative count of DenovoGear probability score (DNMsc) found by both software programs. Each bar represents a cumulative count for number of candidate de novo mutations found at and above that probability score. Generated by code found in Appendix B.

Figure 7 shows the cumulative count for DenovoGear's Probability Score (DNMsc) found by Both software. For candidates found by Both software, DNMsc found 57% of candidates' highest confidence interval (≥ –0.5). Also, like in

Figure 6, there is a constant rise in the number of DNM candidates being counted at each DNMsc threshold, but in a lesser amount than seen in

Figure 6. This means that for DNMsc there is also no clear cut-off where the number of true positives being included outweighs the number of false positives being identified.

For the DNM candidates found by Both software, there was a difference of 59 candidates identified in the top scoring threshold. This difference is on the same candidate DNMs and suggests there is a disparity in sensitivity between the two scoring systems. The more conservative estimates made by PedFilter suggest a high-precision approach, being more selective to avoid false positives. DenovoGear with a more generous count in the top threshold suggests a high-sensitivity approach, trying to catch all true positives but with a higher potential for identifying false positives.

Candidate DNMs found only by PedFilter or DenovoGear

Each software identified several candidate DNMs that were not found by the other software. DenovoGear found a total of 711,259 while PedFilter found 2,305.

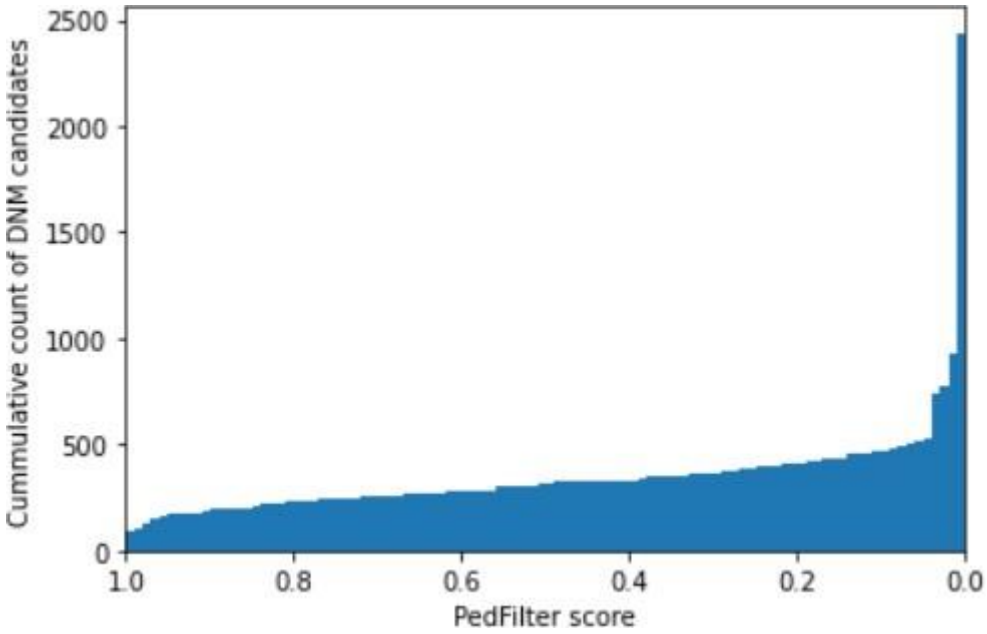


Figure 8: Cumulative count of Pedfilter Denovo-Posterior scores for de novo candidates identified by PedFitler. Each bar is a representation of the number of candidate de novo mutations found at and above that probability score (ie 0.8 range is all candidate DNM with a score of 0.8 or higher). Generated by code found in Appendix C.

Figure 8 shows the cumulative count of Denovo-Posterior scores for all DNM candidates found by PedFilter. This includes the candidates also found by DenovoGear.

Of the total 2,305, only 4.8% of DNM candidates were found in the highest confidence interval (≥ 0.98). Much like the trend seen in

Figure 6, there is a constant increase in DNMs as the score decreases. However, unlike in

Figure 6, Figure 8 shows a peak at 0.05 with 79% of the candidates (1,929) falling in this extremely low range. In this peak, a significant number of the candidates are likely to be false positives, but this cut-off threshold is a lot lower than expected and therefore not a useful suggested threshold cut-off.

This peak also matches the small peak seen in

Figure 6. This suggests that, because there are candidates found by Both software in this range, there is the potential that cutting out DNM candidates at the lower 5% confidence interval could miss some true positives.

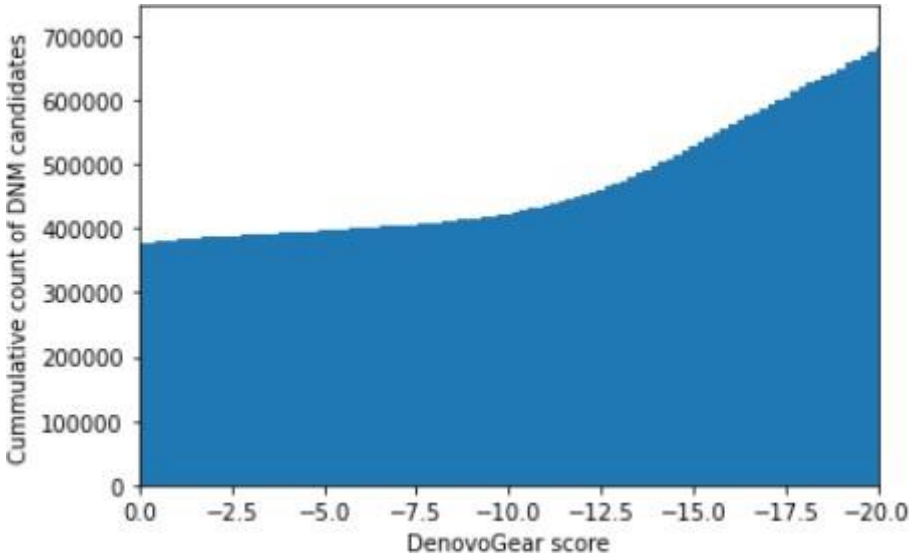


Figure 9: Cumulative count of DenovoGear probability score (DNMsc) for de novo candidates identified by DenovoGear. Each bar represents a cumulative count for the number of candidate de novo mutations found at and above that probability score. Generated by code found in Appendix B.

Figure 9 shows the cumulative count of DNMsc for DNM candidates found by DenovoGear. This includes candidates found by PedFilter as well. DenovoGear found 711,259 DNM candidates, with 53% found in the highest confidence interval (≥ –0.5) and 12.5% of those getting a 0.0 (perfect) score.

The rise in count as the DNMsc drops has a very low growth until the –10 confidence interval before consistently rising at a higher rate. This is a very different shape from the one found in Figure 7. Figure 7 exhibits a more constant rate of increase with the most noticeable jump seen at the –5 confidence interval.

Figure 9 shows a growth pattern closer to exponential growth as the DNMsc decreases. The consistency of growth in

Figure 9 again means DNMsc provides no clear threshold where the number of false positives is likely to outweigh the number of true positives gained. This is consistent with DNM candidate counts in Figure 7.

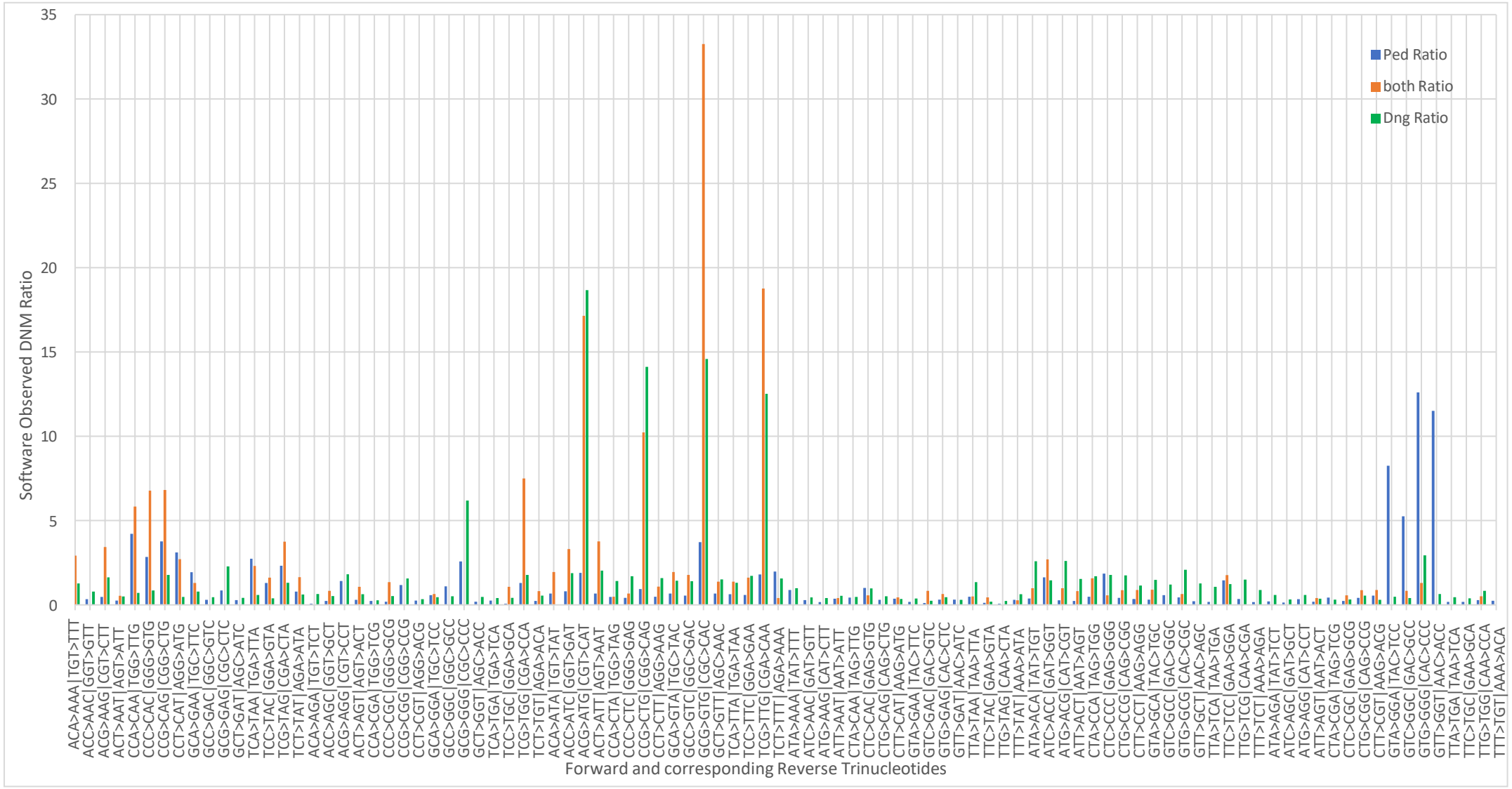


Figure 10: Trinucleotide signatures for DNM candidate ratios observed by PedFilter (Ped, blue bars), DenovoGear (Dng, green bars), and candidates observed by both software packages (both, orange bars). Ratios for PedFilter and DenovoGear include trinucleotide counts for de novo mutations identified by both software packages

Trinucleotide Mutational Signature using Proportional Percentages

Figure 10 shows trinucleotide (trinuc) ratios for DNM candidates grouped by the software that identified them. Trinucleotides are a 3-base sequence. For the DNM candidates, this sequence contains the base at the candidate DNM position, 1 base prior and one base after. This 3-base sequence is selected at the candidate chromosome and position twice: once with the reference allele as the central base, and once with the DNM candidate SNP as the central base. These two trinucleotides are sat side by side to indicate the reference into the mutated trinucleotide and linked to their reverse-read counterparts to make the 96 possible trinucleotide SNP mutations. For example, Ref sequence ACA into alt sequence AAA becomes ACA>AAA and is a C>A SNP mutation. The reverse of this would be TGT>TTT meaning any Trinucleotides ACA>AAA or TGT>TTT are counted and grouped. This data is part of PedFilter output but to get this information for DenovoGear extra processing was required (Appendix A.vi and Appendix B).

In Figure 10 there are four peaks denoted by DenovoGear and Both for TCG>TTG|CGA>CAA, GCG>GTG|CGC>CAC, CCG>CTG|CGG>CAG and ACG>ATC|CGT>CAT. These are all C>T|G>A SNP transition mutations. This pattern matches other Bovine data seen in Harland, 2018 and Harland et al., 2017. This boosts confidence that the data produced by DenovoGear, and those candidates observed by Both, are more likely to be true positives.

The four PedFilter peaks on the right side of the graph are T>G|A>C SNP transversion mutations. These are very unusual as the previous bovine literature (Harland, 2018; Harland et al., 2017) showed that this type of SNP was the least common in each situation they tested. These PedFilter peaks suggest many false positives are being identified. If this is true, then filtering out these SNP types could remove the largest number of false positives with the least number of true positives being lost/affected.

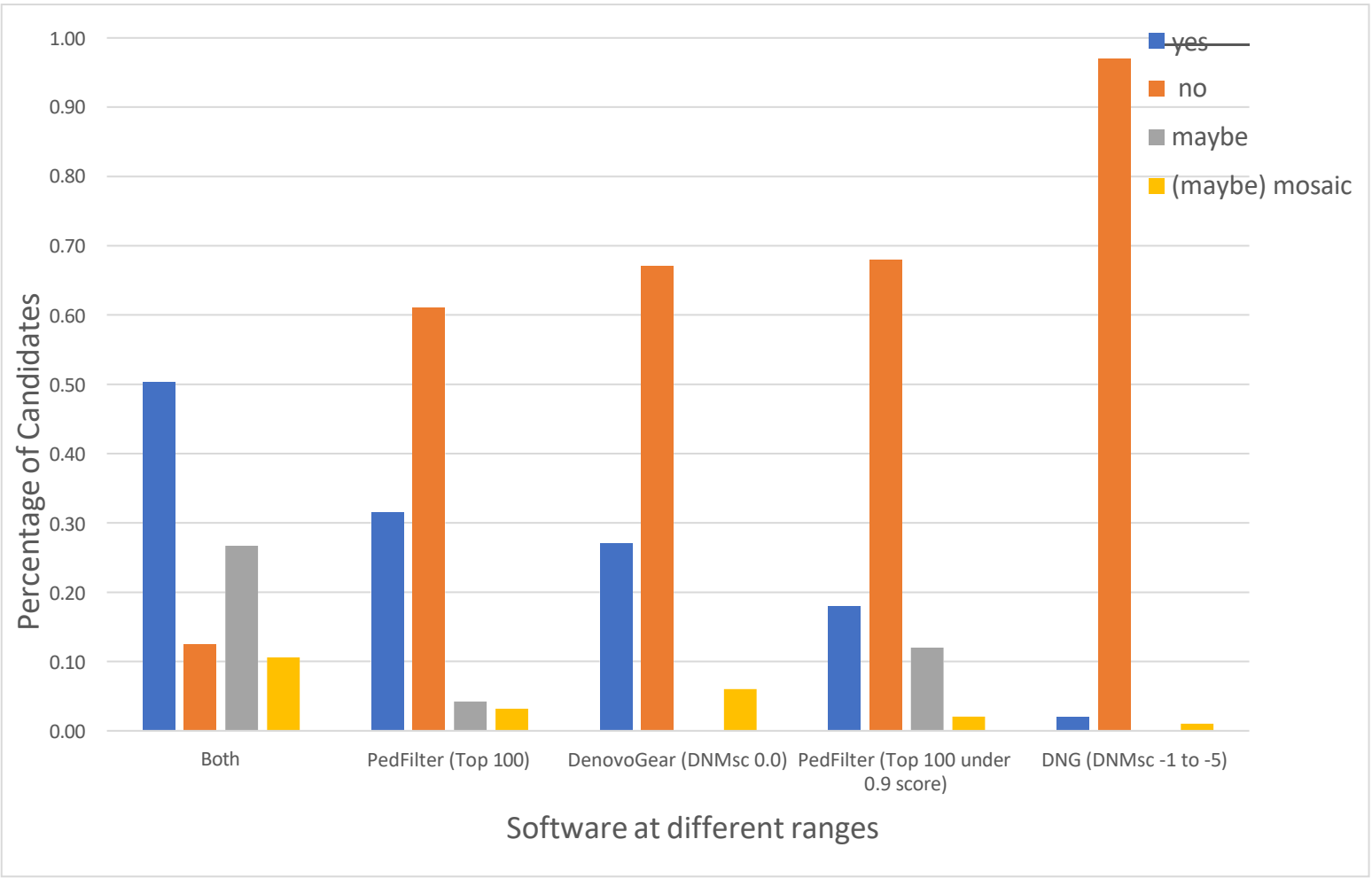


Figure 11: Percentage of verified true and false, maybe and possible mosaic (maybe mosaic) de novo candidate mutations identified by PedFilter, DenovoGear and both packages. Candidates included in the top and lower categories of PedFilter and DenovoGear may include some of the candidates identified by both software packages

Table 8: Percentage of verified true positive, false positive, maybe and (maybe) mosaic of visually assessed de novo candidates

Software	True Positive (Yes)	False Positive (No)	Maybe	(maybe) Mosaic
Both	0.50	0.12	0.27	0.11
PedFilter (Top 100)	0.32	0.61	0.04	0.03
DenovoGear (DNMsc 0.0)	0.27	0.67	0.00	0.06
PedFilter (Top 100 under 0.9 score)	0.18	0.68	0.12	0.02
DNG (DNMsc –1 to –5)	0.02	0.97	0.00	0.01

Figure 11 shows the Percentages of validity given to the IGV visually assessed DNM candidates. Grouped by Score ranges with all candidates identified by Both software were manually IGV verified (161 candidates). The top 100 PedFilter candidates, 100 lower PedFilter (top 100 with a score under 0.9), 100 randomly selected 0.0 scoring DenovoGear candidates, and 100 randomly selected DenovoGear candidates from the –1 to –5 scoring range were also manually IGV verified. The DenovoGear candidates were selected randomly because the sheer number of candidates within a single range made it impractical to manually assess all of them. Also, when ordering by DNMsc to get the top (mostly likely true positives) selection would highly favour one trio's output. The random selection (see Appendix B) allowed for a more even spread across the trios for a more accurate representation of DNM candidates and a more manageable number of candidates to manually assess.

A total of 561 candidate DNMs were manually assessed using IGV (Appendix E, F and G) and assigned a validity tag based on the conditions found in Table 1. Examples of some candidates can be seen in Figure 12-Figure 19.

Figure 11 shows the percentage of candidates validated as yes is highest in Both at 50%. Having only 50% is not ideal as validated true positive candidates are not the majority of those identified. However, this category also has the greatest number of "Maybe" candidates. These maybes are possibly more likely to have a higher percentage be confirmed yes with further testing than the maybe candidates found by only one of the software programs. Analysis of the overall trends, there is a clear decline in the percentage of validated true positives between the top and lower candidates found by only PedFilter or DenovoGear. This decline is met with an equal and opposite rise in the number of false positives (validity tag no). This suggests a correlation between the probability scores from PedFilter and DenovoGear and the validity of each software-identified candidate.

This correlation suggests that while the programs are not 100% accurate yet, their scores show correlation to true and false positive DNM candidates. Further development may provide more accuracy in the future.

Examples of IGV Output and Validation

Yes Examples from Candidates Identified by Both Software Programs

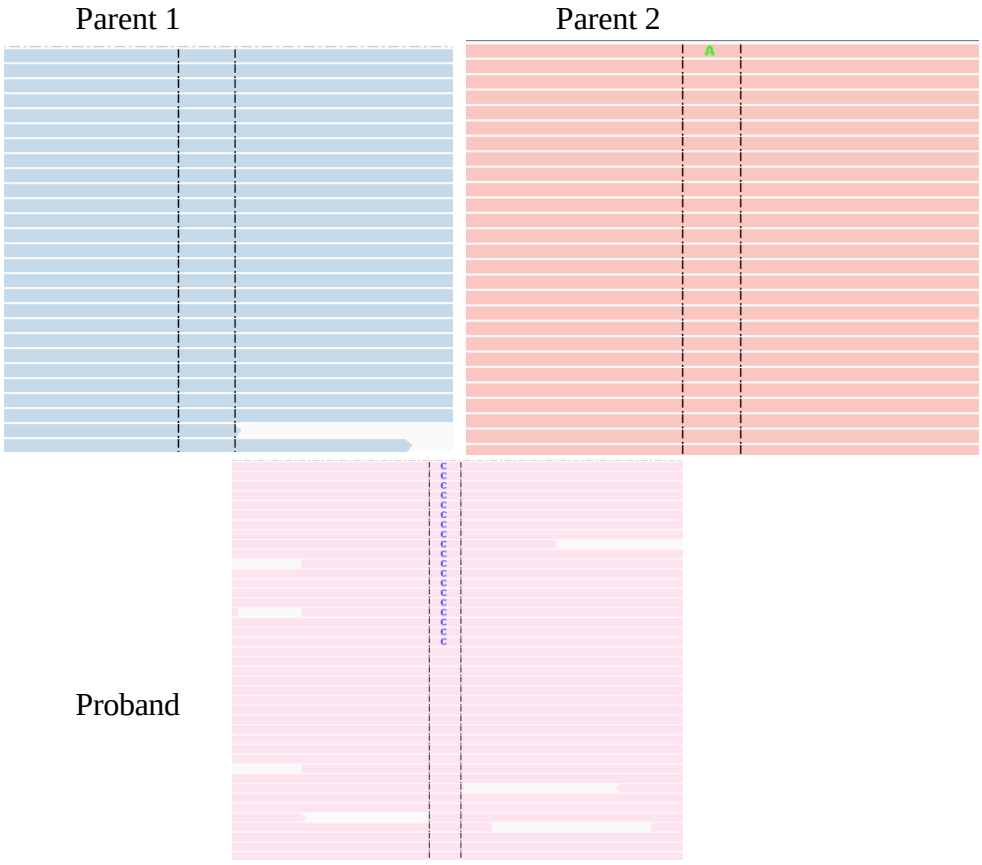


Figure 12: Chrom:29 Position:48366041 Identified by both software packages and validated as Yes

Figure 12 shows DNM candidate Chrom29:Position48366041 in IGV viewer. The reads are grouped and coloured by sample and then ordered by bases. This example has the proband with a significant number of reads (approx. 50%) showing a T>C SNP. This SNP is not seen in either parent but an alternate SNP of T>A is noted on one read in one parent. This suggests that this position has a higher chance of DNM occurring. Not seen in Figure 12 but observed during IGV validation, no unrelated samples from the other tested trios had reads showing any SNPs at this position.

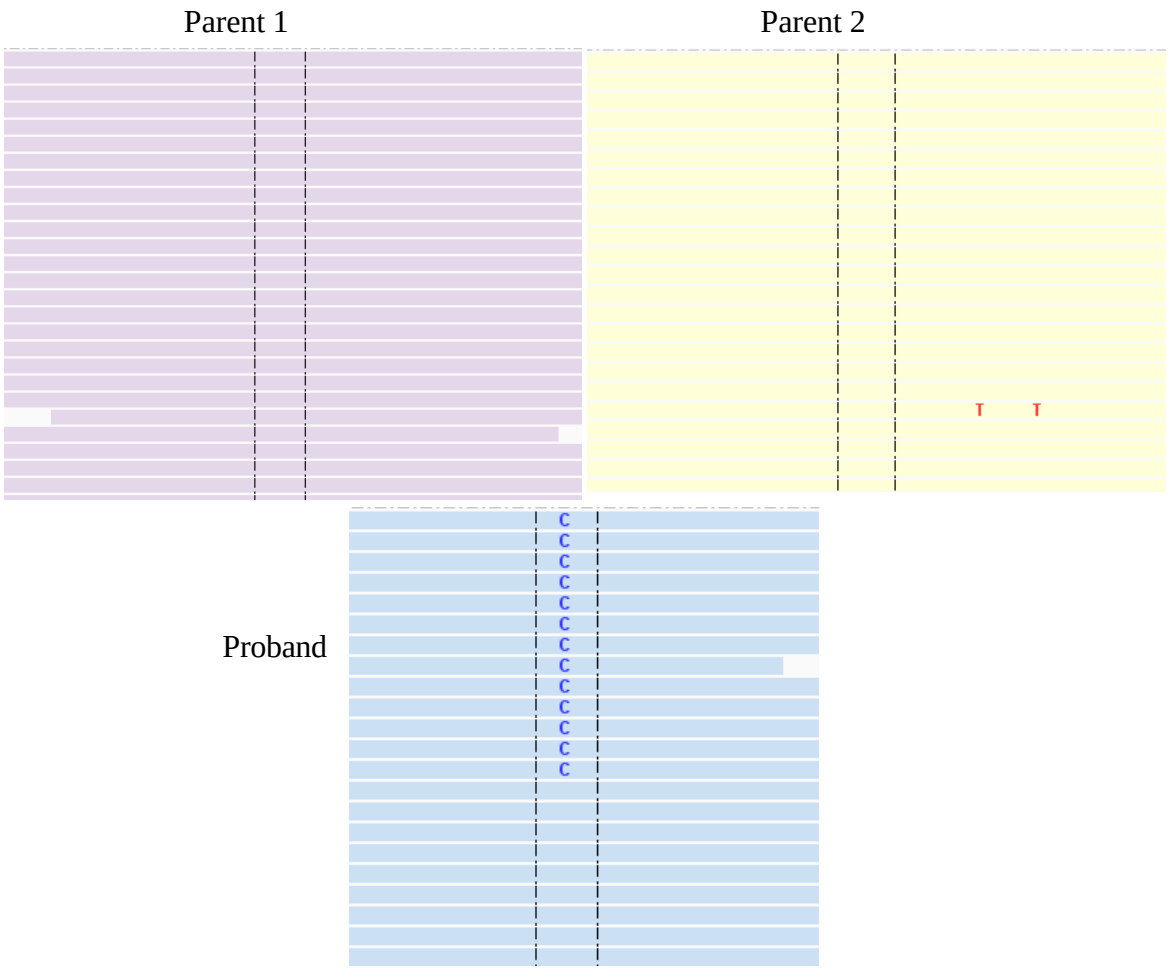


Figure 13: Chrom:20 Position: 65132214 Identified by both software packages and validated as Yes

Figure 13 shows the IGV view of DNM candidate Chrom20:Position65132214. Reads are coloured and grouped as in Figure 12. In this example again approx. Fifty percent of reads contain the DNM-identified SNP which is a T>C. Neither parent shows any SNPs at the read position but parent 2 does show base variation within the read window. This does not affect the validity of the DNM at Chrom20:Position65132214. Again, not seen in

Figure 13 but observed during IGV validation, no unrelated samples showed any SNPs at this position.

No Examples from Candidates Identified by Both Software Programs

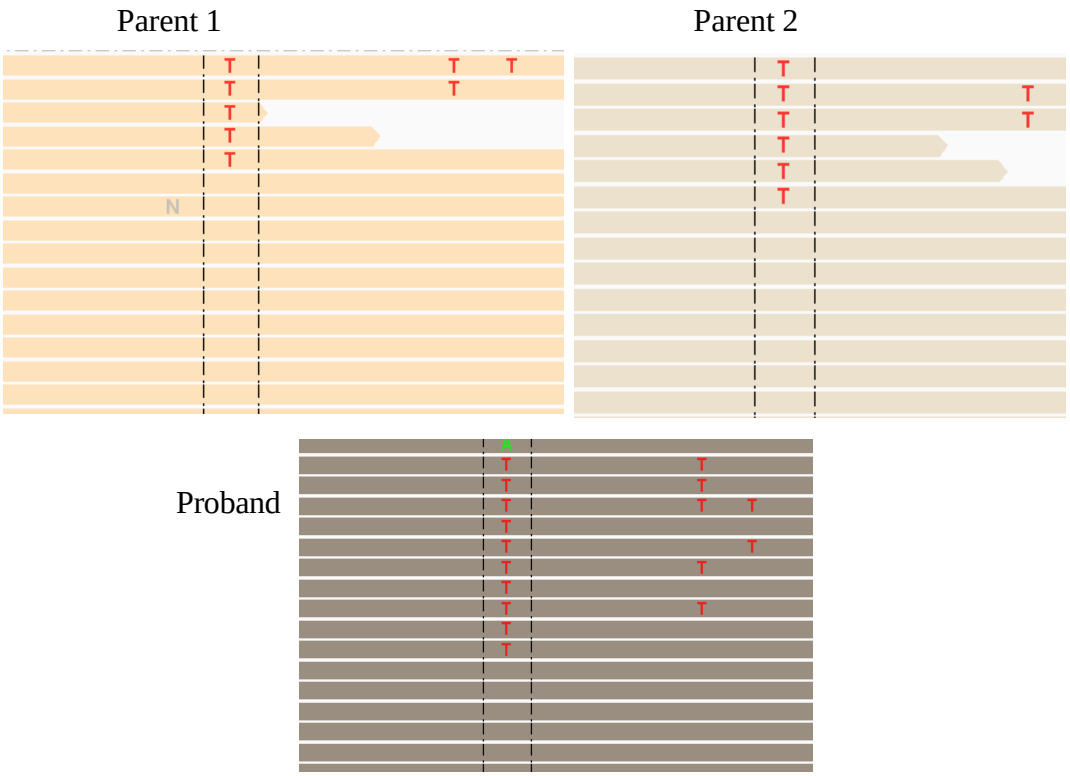


Figure 14: Chrom:10 Position: 66873955 Identified by both software packages and validated as No

Figure 14 shows an IGV view of DNM candidate Chrom10:Position66873955. Reads are coloured and grouped as seen in Figure 12. This example shows multiple reads in both parents as well as a SNP downstream from the DNM candidate. The proband displaying both the DNM candidate and the downstream SNPs seen in both parents (parent 1 SNP is further downstream than parent 2) shows that this is an inherited mutation, not spontaneous, and has been incorrectly flagged. This is also a G>T|C>A mutation. In Harland 2018, this type of SNP was the second most common SNP identified by PedFilter in their study and all those candidates showed signs of Mendelian inheritance. This mutation is therefore indicative of a false positive DNM result from PedFilter which is expected. To also be identified by DenovoGear as a false positive also, means that there is further improvement to both software programs' filtering and identification processes.

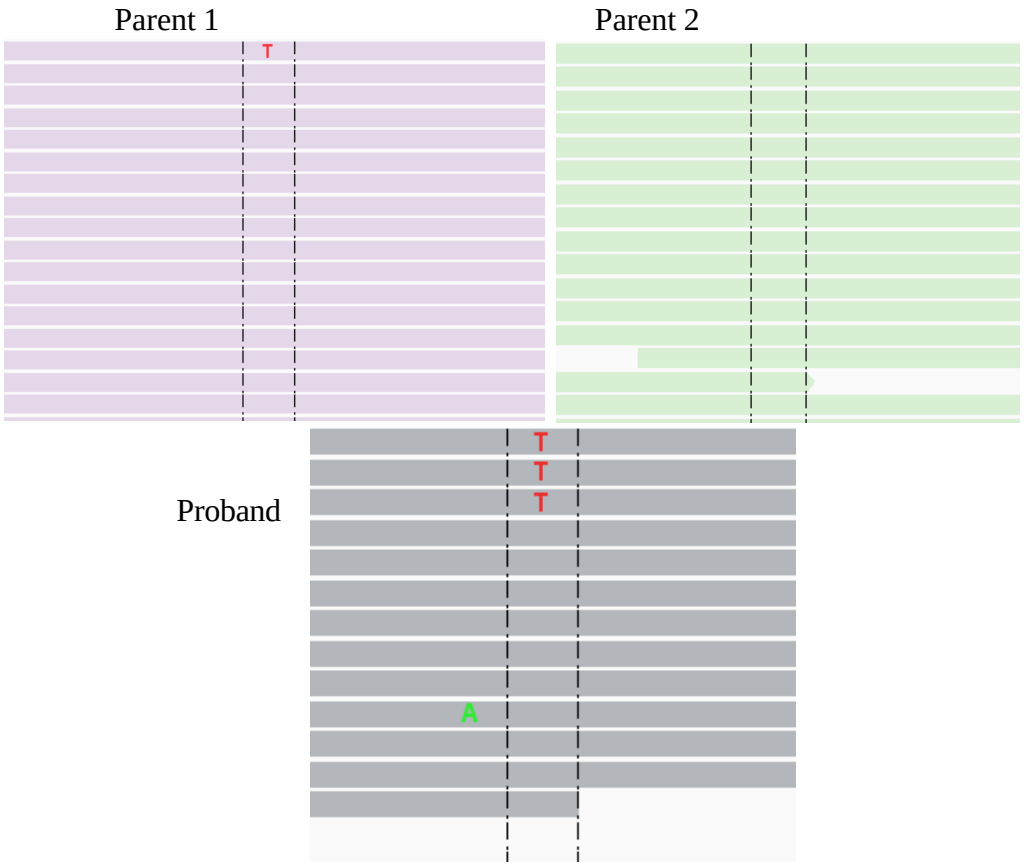


Figure 15: Chrom:25 Position: 11411921 Identified by both software packages and validated as No

Figure 15 is an IGV view of DNM candidate Chrom25:Position11411921. Reads are coloured and grouped the same as in Figure 12. Unlike the No example in Figure 14, only one parent has a single read for the DNM allele. However, the proband only has three Alt reads which equate to approx. 20% of the reads, an insufficient percentage for accurate variant calling, as well as a low total number of reads. This variant is another G>T|C>A SNP mutation, pointing again to the likelihood that this is a false positive candidate.

Maybe Examples from Candidates Identified By Both Software Programs

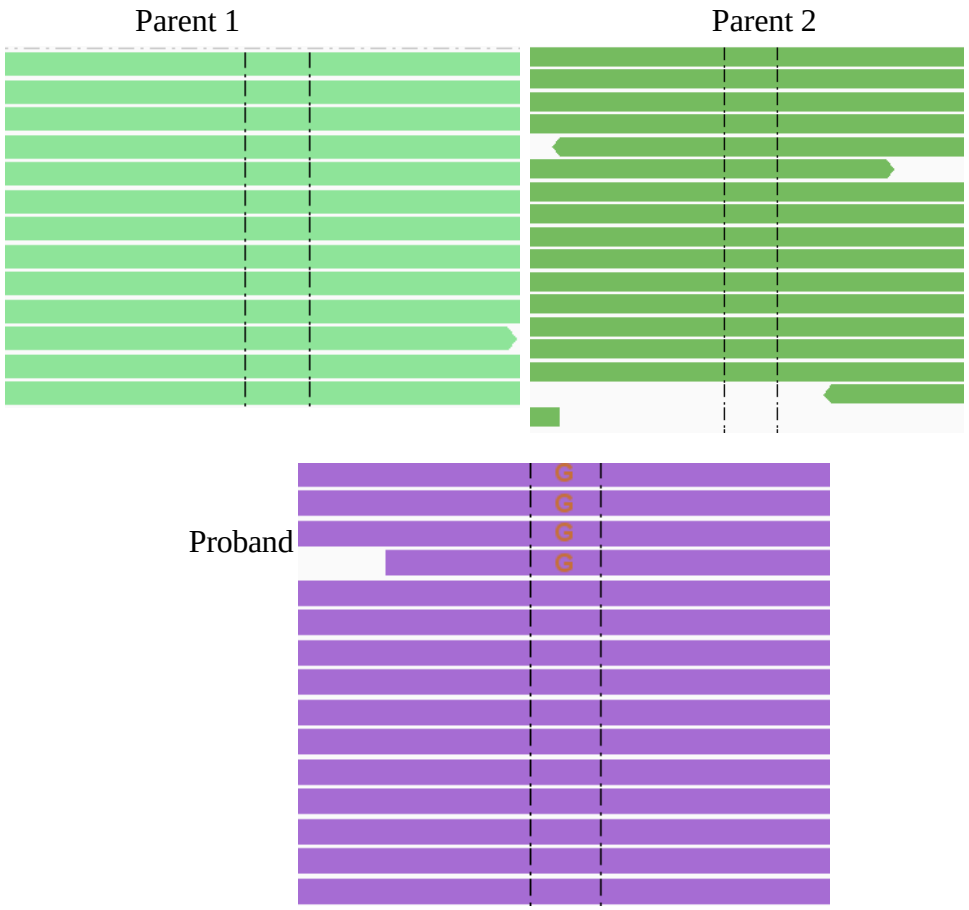


Figure 16: Chrom:9 Position: 46750818 Identified by both software packages and validated as Maybe

Figure 16 is an IGV view of DNM candidate Chrom9:Position4650818. Reads are coloured and grouped the same as in Figure 12. This is an example of a "Maybe DNM candidate". The proband sample has four DNM reads for approx. Twenty percent of the reads. This is similar to Figure 15, on the edge of the required read percentage for no/maybe. However, this mutation has no Alt reads in either parent. Also observed were three unrelated samples with reads for two different alternate alleles. The variety of alternate alleles suggests this region/position could be a high mutation area with a higher potential for this candidate to be a DNM or a region of erroneous reference sequencing leading to false identifications.
The reason it is not a Yes, though, is due to the low total number of reads in the proband and the low percentage of reads with the DNM allele. To confirm the validity, this region would need to be targeted for resequencing at a higher read depth to confirm yes or no. For this research, resequencing was not conducted and this variant, along with other ambiguous DNM candidates identified by PedFilter and DenovoGear, will remain classified as "Maybe."

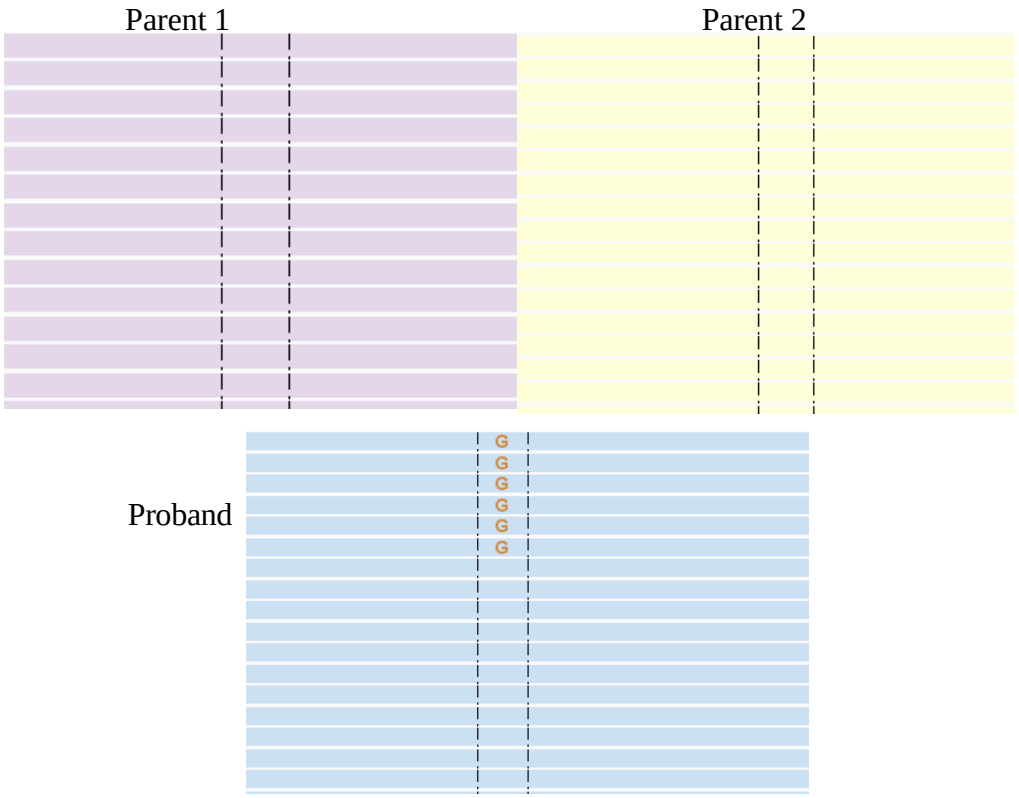


Figure 17: Chrom:18 Position: 38066404 Identified by both software packages and validated as Maybe

Figure 17 shows an IGV view of candidate DNM Chrom18:Position38066404. Reads are coloured and grouped the same as in Figure 12. Due to the total number of reads, this is only a partial view of reads for parents 1, 2 and proband. This “Maybe” example is similar to both Figure 15Figure 16. There are 6 DNM allele reads but due to the total number of reads this is only a lower percentage of approx. 20%. This is on the edge of the threshold percentage between the No and Maybe classifications, but there is a larger total number of reads and a larger number of DNM candidate reads observed than in the sites illustrated in Figure 15Figure 16. There is also one unrelated sample with a single read for a different alternate allele, fewer than seen in Figure 16, and likely has little to no significance to this candidate DNM.

This candidate DNM is T>G|A>C transversion mutation. From the research seen by Harland, 2018 and Harland et al., 2017, this is the class of SNP mutation least likely to be identified by PedFilter. However, while uncommon, it is not completely unexpected or impossible. As above, this DNM candidate would require targeted resequencing for confirmation but again, this was not conducted for this study and this site will remain as part of the ambiguous Maybe class of DNMs identified by the software programs.

Mosaics (Maybe) Examples of Candidates Identified By Both Software Programs

This section shows examples of DNM candidates where one of the proband's parents could be mosaic but would need further testing to prove whether they are or not.

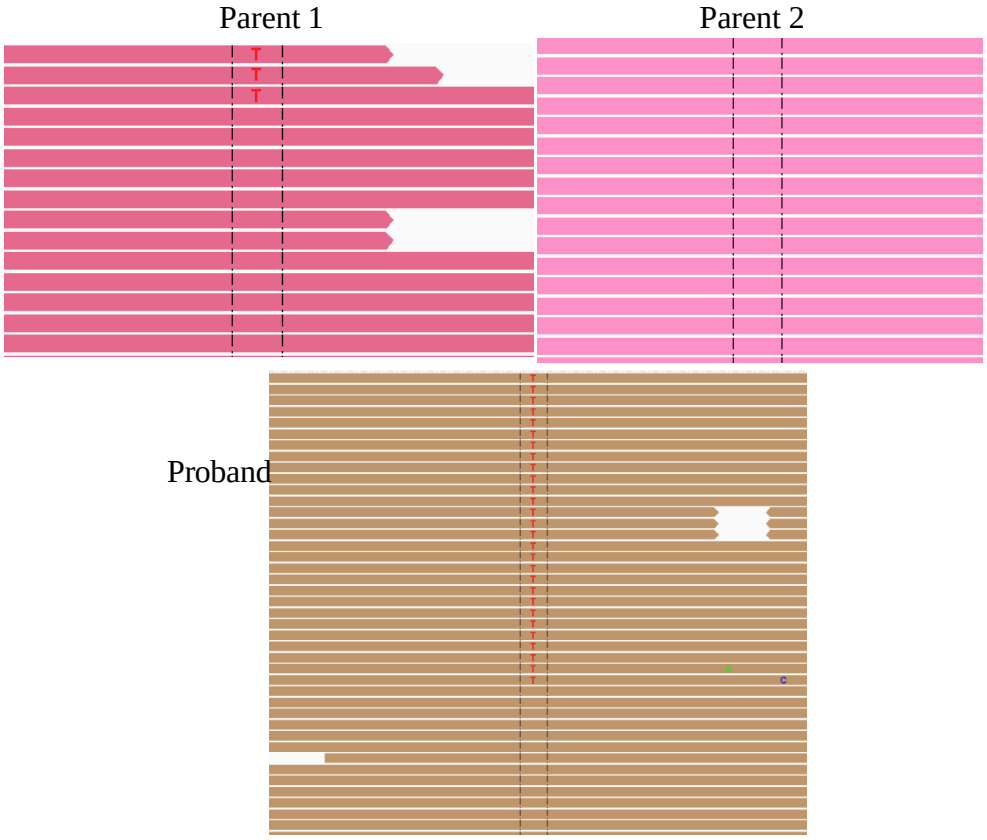


Figure 18: Chrom:6 Position:63530124 Identified by both software packages and validated as Mosaic

Figure 18 shows the IGV view of the DNM candidate Chrom6:Position63530124. Reads are coloured and grouped the same as in Figure 12. Only a subset of the reads are shown for each parent of the proband.

For this DNM candidate, the proband was flagged as having the DNM. They show an approx. 50% number of reads with the alt allele. This high percentage suggests the allele was inherited from a parent. One parent showed 3 reads which is approx. 10% of the total number of reads. This small read percentage is not usually large enough to be identified by DNM software as a candidate on its own, but the combination of the portion of reads found in the proband and one of the parents suggests that this is very likely to be a mosaic.

This is also a G>T|C>A SNP. As noted in Figure 15, Harland, 2018, and Harland et al., 2017, showed this type of SNP mutation followed Mendelian inheritance. This is seen by the nearly even split of T allele from parent 1 vs number of reads matching the reference allele (G) from parent 2.

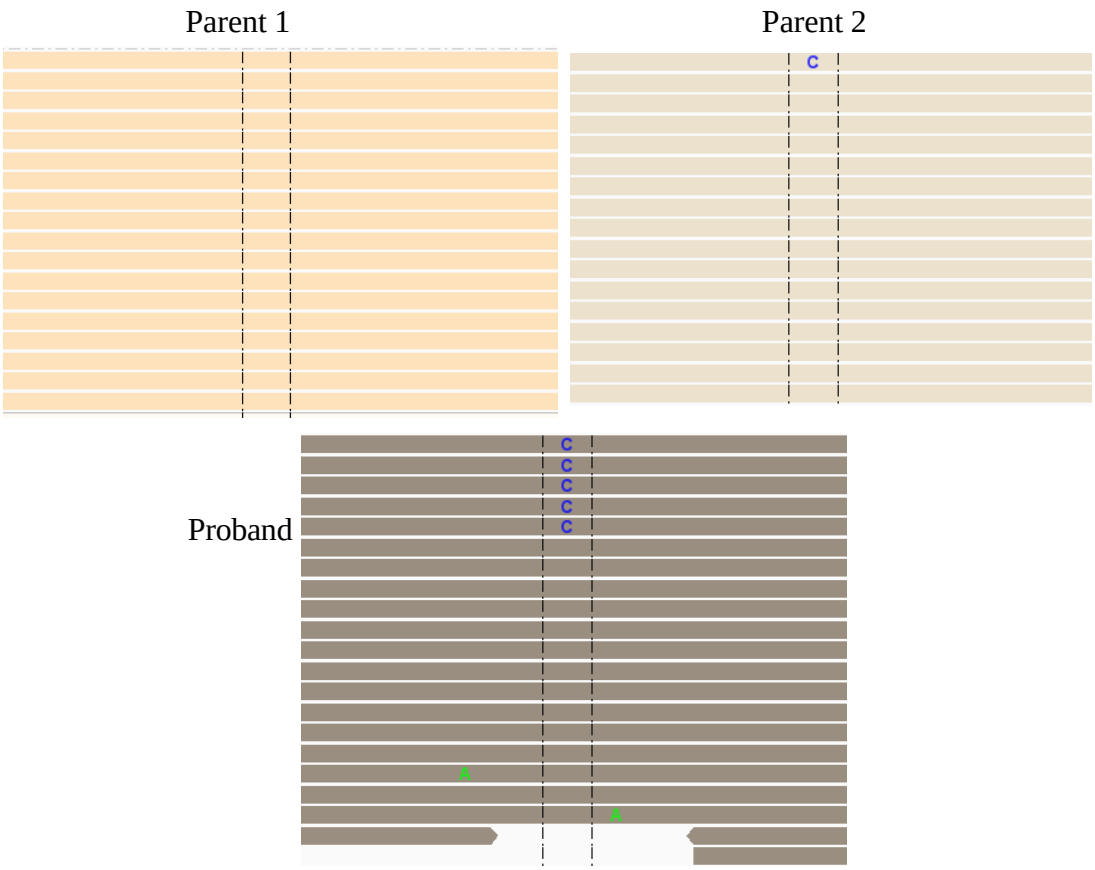


Figure 19: Chrom:4 Position:86057953 Identified by both software packages and validated as "maybe" Mosaic

Figure 19 is an IGV view of the DNM candidate Chrom4:Position86057953. Reads are coloured and grouped the same as in Figure 12. This candidate is validated as Maybe mosaic, different from Figure 18, and needs resequencing to confirm but is very likely a mosaic. The DNM candidate found in Figure 19 only has a read percentage of approx. 25%. This is similar to the candidates that were classed as Maybe after visual assessment but unlike those candidates, Figure 19 has one parent display a single read.

Also observed at this candidate site were three unrelated samples with this SNP and two unrelated samples with a different alternate SNP. This could suggest a potential hotspot for mutation, but the number of reads observed in the proband, parent and unrelated samples are all low (low percentages). This could also suggest a hard-to-sequence location and a systematic or sequencing error occurred leading to the single reads in the parent and unrelated samples.

The uncertainty surrounding this candidate means a definitive validity tag (yes/no/mosaic) would require targeted resequencing like the other ambiguous (maybe) candidates. The possibility of the candidate being a mosaic means it is included in the mosaic statistics but lacking the clarity seen in Figure 18, is classed differently as maybe mosaic

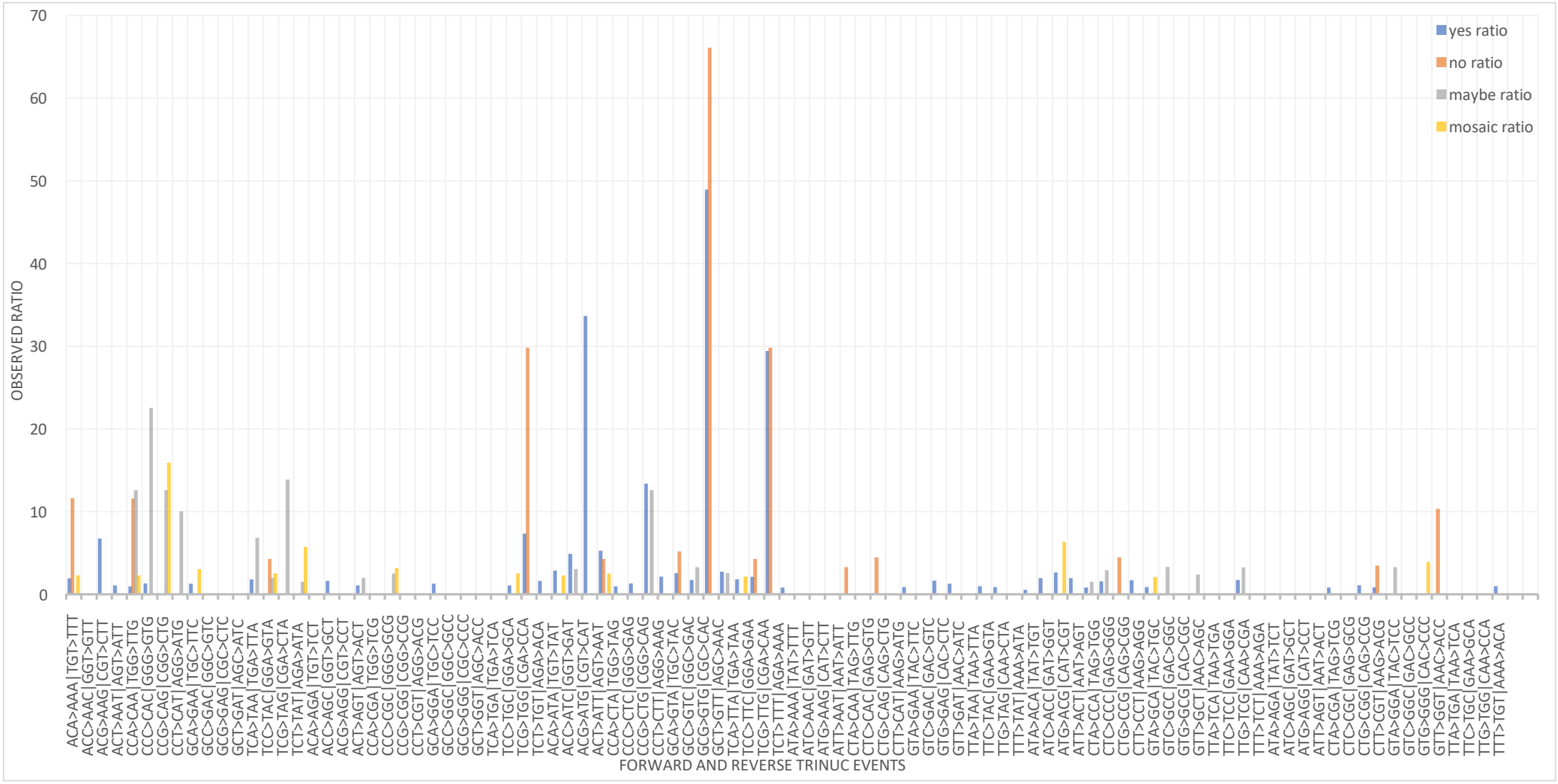


Figure 20: Trinucleotide mutation signature for de novo candidates identified by both PedFilter and DenovoGear. Data is a proportional percentage calculated using the observed ratio, expected genome ratio and total count of trinucleotide. Data is grouped by validity tags for true positive (yes), false positive (no), ambiguous (maybe) and possible mosaics (mosaic)

Trinucleotide Proportional Percentages for DNM Candidates Identified by Both Software programs PedFilter and DenovoGear

Figure 20 shows the different trinucleotide ratios for validated candidates found by Both PedFilter and DenovoGear. Similar to Figure 10Figure 11, Figure 20 has four peaks. Going left to right those peaks are TCG>TGG, ACG>ATG, GCG>GTG, and TCG>TTG and their reverse read counterparts. The C>T peaks are expected based on prior research (Harland, 2018; Harland et al., 2017). However, the similar or high proportion of false positive to true positive DNM candidates seen at some of these peaks (GCG>GTG and TCG>TTG) is unusual and does not fit within the expected true positive C>T candidates.

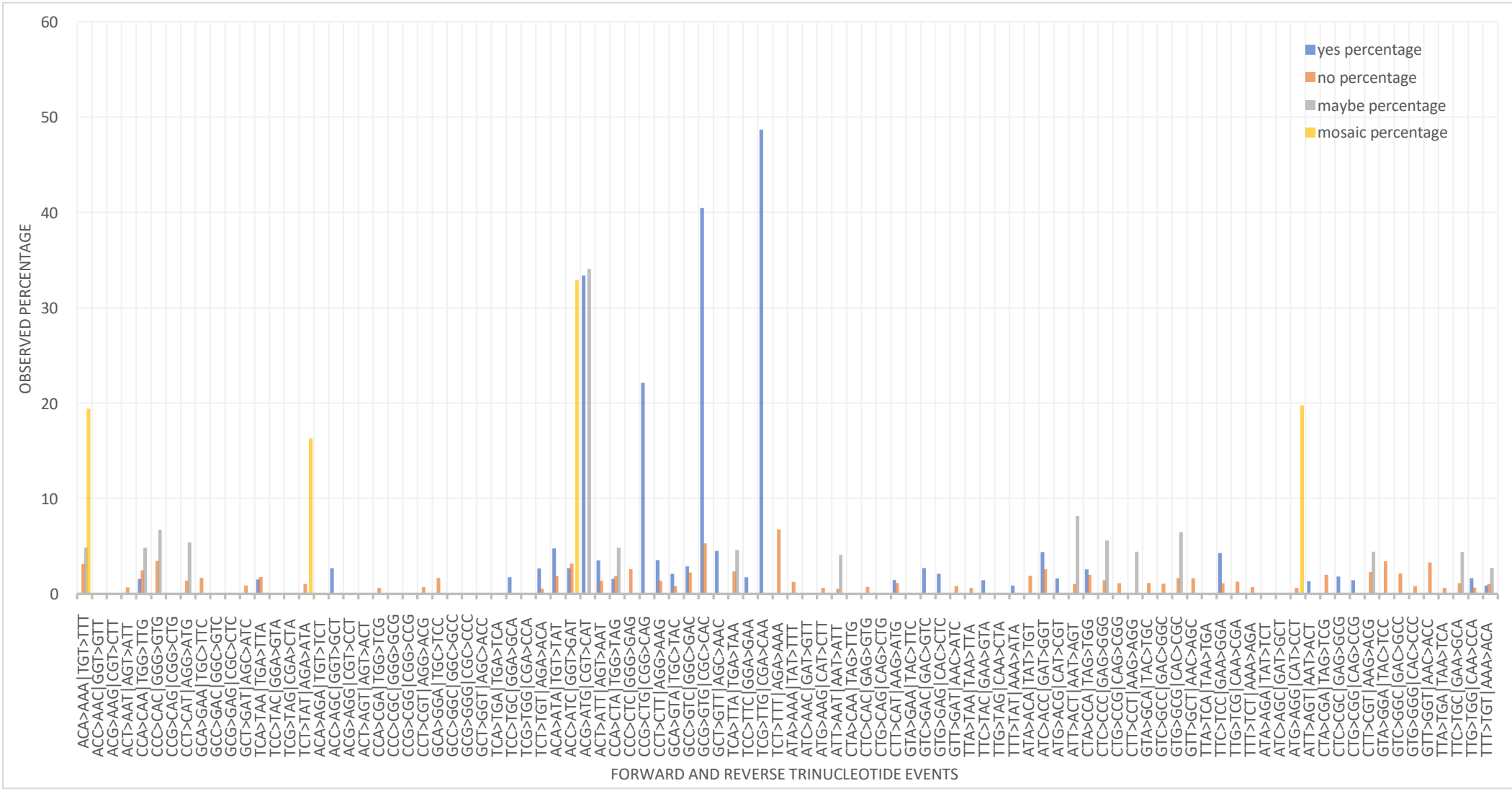


Figure 21: Trinucleotide mutation signature for top 100 de novo candidates and 100 de novo candidates with a probability score of less than 0.9 identified by PedFilter. Data is a proportional percentage calculated using the observed ratio, expected genome ratio and total count of trinucleotide. Data is grouped by validity tags for true positive (yes), false positive (no), ambiguous (maybe) and possible mosaics (mosaic)

Trinucleotide Proportional Percentages for 100 top and 100 lower scoring DNM Candidates Identified by PedFilter

Figure 21 shows the mutational signature for trinucleotides of the top 100 DNM and 100 DNM candidates with a probability score of less than 0.9 identified by PedFilter.

Table 8: Percentage of verified true positive, false positive, maybe and (maybe) mosaic of visually assessed de novo candidate shows both the top 100 and lower 100 PedFilter candidates have less accuracy than that seen in candidates identified by both software packages. However, the mutation signature seen in Figure 21 more closely follows the expected output outlined in Harland, 2017 with less noise from false positive peaks.

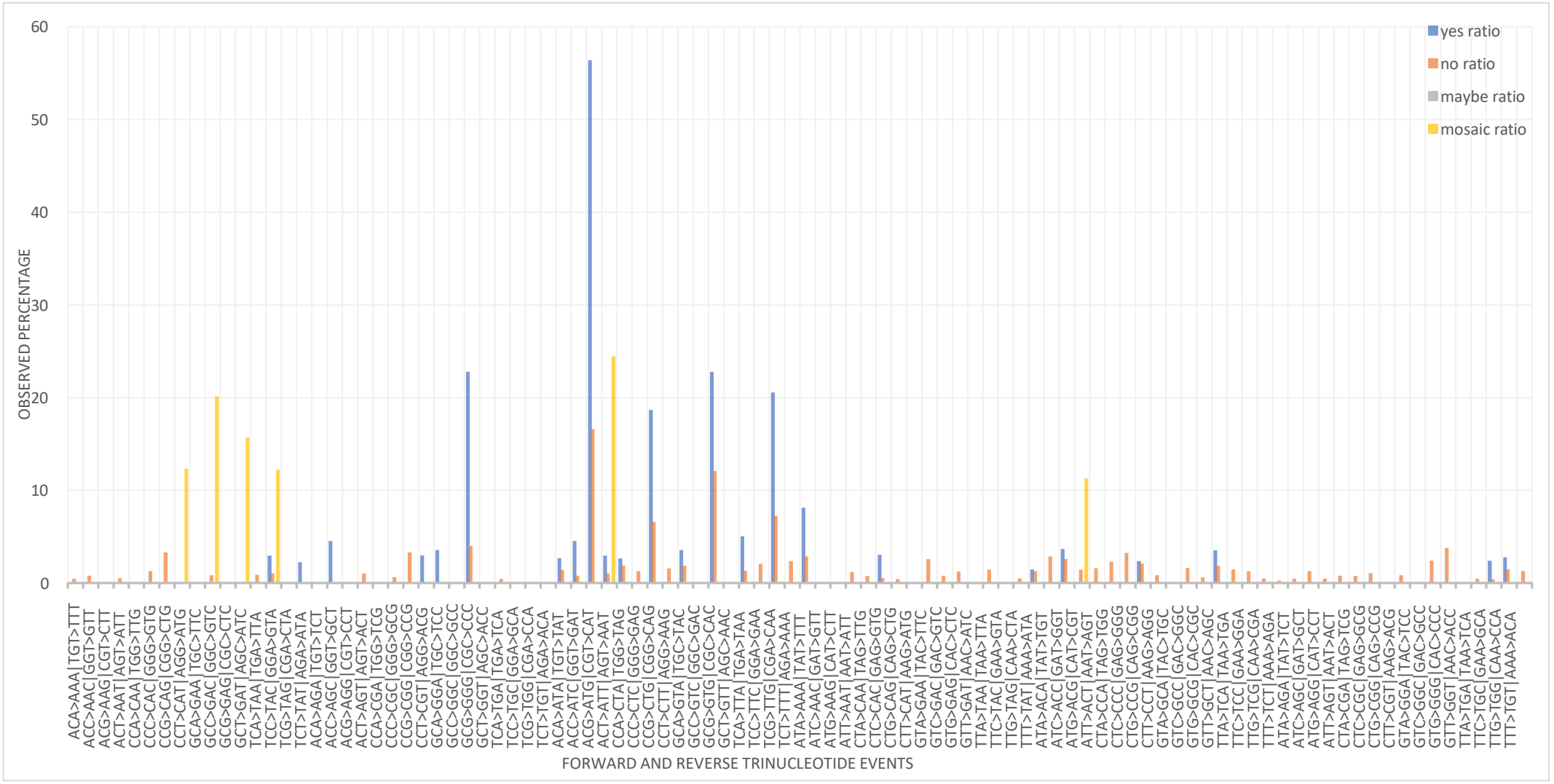


Figure 22: Trinucleotide mutation signature for 100, 0.0 scoring de novo candidates identified by DenovoGear and 100 lower-scoring candidates from the score range -1 to -5. There are more than 100 DenovoGear DNM candidates for each range (0.0 and -1 to-5). These 200 candidates are a random selection of candidates spread across all six of the trios as a representation of these score ranges for DenovoGear candidates. Data is a proportional percentage calculated using observed ratio, expected genome ratio and total count of trinucleotide. Data is grouped by validity tags for true positive (yes), false positive (no), ambiguous (maybe) and possible mosaics (mosaic)

Trinucleotide Proportional Percentages for 100 high Scoring DenovoGear DNM candidates and 100 lower scoring candidates

Figure 22 shows the mutation pattern for all DenovoGear DNM candidates that were evaluated. For this set, a random selection of 100, with a score of 0.0 and 100 candidates with a score between -1 and -5 were selected from the six trios.

Figure 22 has five peaks, four of which fall within the C>T transition space of trinucleotides. Like the PedFilter data, there are no significant false positive peaks. However, unlike the PedFilter data, all the DenovoGear true positive peaks have a corresponding lesser false positive peak. This can probably be attributed to the significant difference in the number of candidates identified by DenovoGear compared to PedFilter. The larger number of candidates identified by DenovoGear likely lead to a higher chance of false positive candidates being identified and therefore more likely to be selected for IGV assessment.

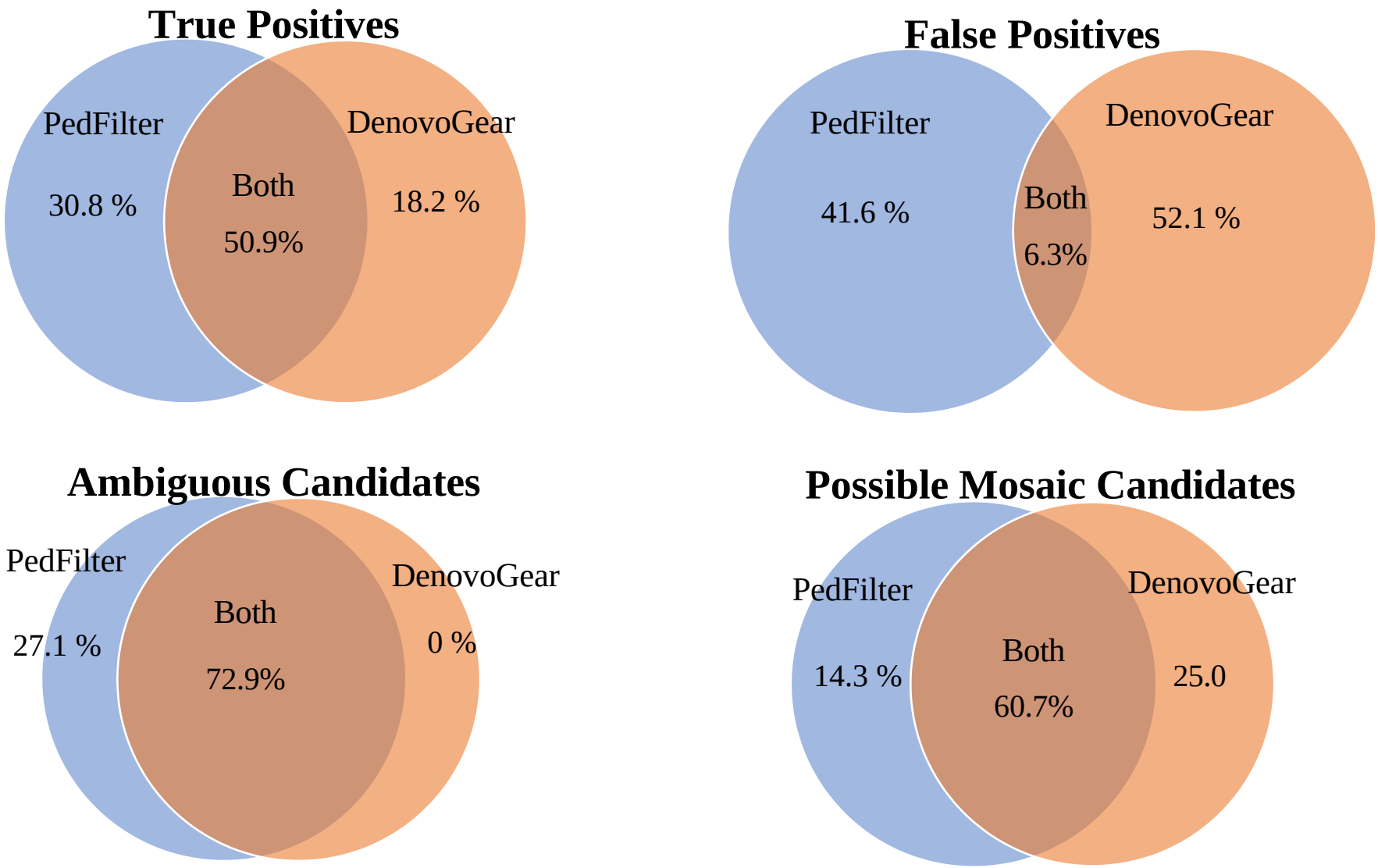


Figure 23: Venn diagrams showing percentage of candidates in each assessment category found by either PedFilter, DenovoGear or both. The total count of each category (True positive, false positive, Ambiguous and possible Mosaic candidates) was separated out and percentage found by each software package was calculated.

Percentage of IGV assessed categories found by PedFilter, DenovoGear and Both

Figure 23 contains a series of Venn diagrams outlining the percentage of true positive, false positive, ambiguous, and potential Mosaic candidates as categorised by IGV visual assessment found by either PedFilter, DenovoGear or both. This was calculated using the total number of assessed candidates assigned to the category (ie all the yes/true positive candidates) and the number of candidates found by each or both software packages.

Figure 23 shows for every category, except for false positives, the largest percentage was found by both software packages. This is to be expected as it is unlikely that both software packages find the same false positives and that both software packages found the most likely candidates.

Chapter 4: Discussion

The Problem

The rapid development of NGS technologies in recent years has led to a significant expansion in genetic and bioinformatic research. With this expansion, many new programs and pipelines have been created to accommodate the efflux of big data output. For newer areas of bioinformatic research like the pursuit of DNM identification, there are not many guidelines or existing protocols designed to ensure standards of research are high. The lack of accepted standards for this area has led to different methods implementing a range of ideologies to create the software used to detect DNMs. The purpose of this study was to compare four existing DNM software programs and outline some possible recommendations for future use. By using existing programs, time and resources can be saved and increased use of the programs could potentially encourage further development in them.

Through the course of this work, two DNM-identifying software programs, PedFilter (Harland, 2018) and DenovoGear (Ramu et al., 2013) were run on six bovine trios. Output data from PedFilter and DenovoGear identified possible (candidate) DNMs which underwent several filtering and analysis processes, as outlined in Chapter 2. The results of this testing were displayed in the results chapter with an analysis and discussion to follow.

Two additional programs were also investigated, DenovoCNN (Khazeeva et al., 2022) and RUFUS (Farrell, 2014) but they proved challenging to install and use and were removed from testing. RUFUS was difficult to install onto NeSI, with many dependencies requiring fixed versions, as well as having highly specific requirements to run. Combined with NeSI's strict installation protocols and software version controls, this resulted in the inability to install and run RUFUS correctly. DenovoCNN also struggled with NeSI's strict installation rules. NeSI support was needed to install DenovoCNN as a container. Due to security reasons, full access to the container to manipulate run parameters and input data was not given. This made making adjustments difficult and required assistance from the support team for every update/change to the run. Also, during the process of testing and updating, the DenovoCNN git repository was deleted and moved to a new account. This required a full reinstallation of the software package which would have restarted the testing and setup process. With the difficulty of use and a

required restart to the process, it was decided that DenovoCNN was too time-consuming and it was therefore its use as a DNM-identifying software program ended for this project.

Analysis of DNM Candidates Identified by Both Software Programs

Of the total 714,393 candidates identified across PedFilter and DenovoGear, only 161 of them were identified by both programs. As seen in Table 2, after IGV validation, 50% of these candidates were deemed to be true positives while only 12% were deemed false positives. The remaining 38% of the candidates are split between possible mosaic candidates (11%) and ambiguous candidates that would need targeted resequencing to validate their DNM status (27%). Of the candidates visually assessed using IGV, the candidates identified by both programs had the highest proportional percentage of true positives. This result is reinforced by the fact that the trinucleotide signature for these variants matches that observed for DNM in a previous study (Harland 2018), with the four most frequent DNMs all fitting the pattern NCG>NTG.

Interestingly, with epigenetic signatures, the methylation of the C nucleotide in the CG leads to a higher mutation rate, making these trinucleotides the most common sites for mutations (Zhou et al., 2020). This is because the cytosines within CG dinucleotides are often methylated but are easily deaminated. The deamination of 5-methylcytosine results in thymidine (Fryxell & Moon, 2005). This is why a large number of spontaneous C>T can occur. By contrast, unmethylated cytosine is deaminated into uracil, which is more easily identified in DNA, allowing the mutation to be repaired, resulting in a lower rate of C>T mutations outside NCG sites.

Each true positive peak in Figure 20 is at least 10× higher than the expected genome mutation rate; this is a typical observation for germline DNMs.

The second peak with trinucleotide ACG>ATG fits into the expected C>T mutation class with all candidates identified by Both software programs validated as true positives. Peaks three and four, trinucleotides GCG>GTG and TCG>TTG, are very interesting as they also fit within the expected C>T SNP mutation. Still, there is almost an equal proportion of candidates (peak four) or more (peak three) candidates that were verified as false positives than true positives. This could indicate a software bias in identification for this type of transition SNP or sequencing errors in the parents or proband leading to false identification.

In saying this, the software bias is unlikely to have originated from PedFilter or DenovoGear. Both software packages take in vcf files of variants identified by a variant caller. The variant

caller assigns genotypes for every possible variation from the reference sequence. Both PedFilter and DenovoGear use these genotypes, among other details, to predict the probability that any of the variants are DNM or not. Neither PedFilter nor DenovoGear changes parameters or identification rulings mid-run, therefore the bias seen most likely occurred during the variant calling stage.

Based on research by Harland, 2018 and Harland et al., 2017, the C>G transversion (TCG>TGG trinucleotide peak) SNP is one of the least likely DNM SNPs to occur. The large ratio of these candidates being verified as false positives in this study concurs with those findings. However, the fact that both programs flag a significant number of candidates with the same trinucleotide suggests something is going on here that needs further investigation.

PedFilter Analysis

The output data for the 200 IGV-assessed PedFilter candidates fell within the expected results based on Harland, 2017. As seen in Figure 21, four substantial true positive peaks fall within trinucleotides with a C>T transition. These are ACG>ATG, CCG>CTG, GCG>GTG and TCG>TTG, and their reverse read counterparts. These are ACG>ATG, CCG>CTG, GCG>GTG and TCG>TTG, and their reverse read counterparts. These four peaks have been noted by the Catalogue of Somatic Mutations in Cancer (COSMIC) in signature SBS1 (*COSMIC | Mutational Signatures*, n.d.; Nik-Zainal et al., 2012). The proposed cause of this signature is spontaneous or enzymatic deamination of 5-methylcytosine which results in C>T mismatches. This type of signature is therefore considered ‘normal’ for spontaneous germline mutations. A study performed by Alexandrov et al., 2013, see this mutational signature again in 25 of their 30 tested human cancer types. Alexandrov et al., 2013 suggest that this mutation mechanism operates in the germline and results in significant depletion of CpG sequences in normal cells causing the NCG>NTG peaks of this signature.

Two of these peaks, ACG>ATG and TCG>TTG, also match two of the four peaks in Figure 20. The ACG>ATG peaks in Figure 20 and Figure 21 are similar in size but while the peak seen in Figure 20 has no other validity tags for this trinucleotide, Figure 21 shares the trinucleotide with a similarly sized maybe peak. Due to the similar proportion of true positives in both Figure 20 and Figure 21with this trinucleotide, the ambiguous candidates with this trinucleotide are

potentially more likely to become true positives following targeted resequencing. The ambiguity could also be due to sequencing or variant calling errors leading to a string of false positive identification by PedFilter.

The second shared peak is the TCG>TTG mutation, the highest overall peak for Figure 21. Unlike Figure 20, where the true positive peak for TCG>TTG is matched by false positives, the peak in Figure 21 is twice the size, and all identified candidates were validated as true positives.

Other interesting points in Figure 21 are the four mosaic peaks ACA>AAA, TCT>TAT and ACC>ATC and ATG>AGG. Two of these peaks fall within the C>A SNP mutations. This type of mutation was reported by Harland, 2018 as showing mendelian inheritance. This suggests that one of the proband's parents was mosaic with this SNP and it wasn't until the proband inherited it that it was picked up as a potential DNM. The ACC>ATC peak is the largest mosaic peak and falls within the expected C>T mutation class. This makes it a likely trinucleotide site to undergo DNM and the potential mosaicism masking the identification of it in the original generation with detection happening in the offspring.

The inability of PedFilter to identify or distinguish between mosaic DNM probands and non-mosaic DNM probands was likely due to systematic errors in variant calling, leading to probability scores that inaccurately represent these DNMs. This can cause DNMs to either be missed or invalidated when they do not match expected Mendelian inheritance patterns or follow incorrect pedigrees to track specific DNMs. While it is easy to blame software packages for these flaws it should be noted that due to complexity, to confirm mosaicism within a variant, targeted resequencing is required of both the parents and the offspring.

Of the 200 assessed candidates shown in Figure 21, there is a noticeable lull in candidates with C>G transversion mutations. Harland, 2018 noted that this was one of the least likely types of DNM so finding candidates to fit in these trinucleotides is expected to be unlikely.

Finally, Figure 21 has no notable peaks of false positive DNM candidates, as the false positives are spread out across most of the 96 trinucleotides, with all percentages under 7%.

Overall, PedFilter's true positive candidates fell within the expected output from previous studies and offered a filtered-down version of possible DNM candidates from the total variant list.

DenovoGear Analysis

Unlike PedFilter which extracts possible candidates from the provided list of variants, DenovoGear assigns a probability score to all provided variants, requiring post-run filtering to sort possible/likely candidates from those unlikely to be DNMs. This has led to a vast increase in the number of potential candidates compared to that found by PedFilter.

A total of 89,201 DNM candidates were identified by DenovoGear with log-scale probability scores of 0.0 (i.e., 100% confident). However, a single trio from the study accounted for 88,585 of these, a disparity that was also seen in the number of DNM candidates in the -1 to -5 score range, where 8,357 of the 13,956 candidates found were from that same single trio. It is unclear where the disproportionately large number of candidates came from for this specific trio, as the same pattern is not seen in the PedFilter data. One possible reason is systematic sequencing errors. Many of the false positives observed in IGV for this trio occurred in read windows where a large number of read errors had occurred in the proband, the parents and other unrelated individuals. When looking purely at the variant site and solely at the trio in question, the false positive candidates could be considered DNMs. However, when analysing broader than the immediate trio it becomes obvious these sites are not DNMs and are likely the products of sequencing or mapping errors, or of already existing mutations in the wider population.

DenovoGear's method of investigating information provided by the trio works well for small in-depth studies, such as occur in rare human disease research, that have extremely high-quality sequence data. DenovoGear requires this high-quality input data because it has no way of internally error-checking the validity of the variant file it was given.

The average expected number of DNMs per individual in humans is 30–80 (Acuna-Hidalgo et al., 2016; Goldmann et al., 2021). Estimates for the average number of DNMs in cattle from previous papers are 60–80 per individual for cows aged 6–10 years, and more recent research using younger (2–4 years) New Zealand cattle suggests 20–40 per individual (via private correspondence). Given these estimates, the most likely explanation for the exceptional number of candidates identified by DenovoGear is an error with the sequence data or the variant caller as all the other trios run with DenovoGear fell within or close to the expected number of DNM for New Zealand cattle. Despite this absurd total number of candidates being identified, based on visual observations and the percentage of the genome that was considered for these trios an

estimate between 5.68 to 37.20 DNMs per individual per generation was made. It was is also noted that all but one of the six trios fell within the 20-40 estimate previously discussed.

For a fair analysis of each trio, candidate selection for IGV assessment included an even number of randomly selected 0.0 and -1 to -5 scoring candidates from each trio. These candidates are then used for the DenovoGear mutational signature analysis seen in Figure 22.
Figure 22 shows five notable peaks for true positive candidates. They are for trinucleotides GCG>GGG, ACG>ATG, CCG>CTG, GCG>GTG and TCG>TTG. Trinucleotide peaks ACG>ATG and TCG>TTG are also seen in the PedFilter data. This supports that candidates found with these trinucleotides are likely to be true positives. Two more of the peaks are C>T mutations. According to Kong et al., 2012, two-thirds of their identified and confirmed DNMs were transitions (C>T|T>C and A>G|G>A). This supports Harland, 2018's findings of C>T|T>C being one of the most common DNM SNPS. From this, trinucleotides containing C>T|T>C SNPs are the most likely to be true DNMs and should also be the most commonly identified candidates by both software packages.

Overall Observations

A direct comparison between the two software packages PedFilter and DenovoGear has shown that PedFilter is slightly more accurate when working with bovine trios. This can be seen in Figure 23 where a greater percentage of true positive DNMs were found by PedFilter along with a smaller percentage of false positive candidates identified. However, PedFilter did not identify as many candidates as DenovoGear, suggesting that PedFilter takes a more conservative, high-precision approach as opposed to DenovoGear's emphasis on high-sensitivity identification. PedFilter's high-precision style of identification requires less filtering of false positive candidates, but it is also more likely to miss some of the weaker true positive candidates.

DenovoGear's high-sensitivity approach can be seen in the larger false positive percentage of candidates identified but it also contributed to the higher percentage of mosaic candidates identified. Mosaic candidates represent one of the biggest challenges with DNM research and it is important to identify and classify them to have an accurate understanding of the impact of these DNMs. It is particularly important for breeding programs, such as those run in NZ and

around the world for dairy cattle, to know where a DNM mutation originated, giving the ability to properly mitigate deleterious effects or capitalise on beneficial ones, like the slick gene for more heat tolerant cattle (Davis et al., 2017). DenovoGear's high-sensitivity approach will potentially identify more mosaics but at the cost of also identifying more false-positive candidates.

Beyond both PedFilter's and DenovoGear's different algorithms and emphasis on precision to sensitivity, the quality and read depth of the initial sequence data play a large role in the accuracy of DNM identification. Among the trio samples provided by LIC, some sequences were older and, due to a number of possibilities might be less even in their genome coverage with low read depth in AT or GC rich regions. Some factors include the cost of sequencing at the time, being run on different sequencing machines and or under different protocols, leading to different rounds (years) of sequencing having slightly different genome coverage.

There was a minimum average read depth requirement of 20 that all of the individuals from each trio met; however, an average read depth of 20 meant there was a potential for large portions of the DNA sequence to be less or more than this.

For some of the older sequences, there were significant regions where the read depth was less than 20, which led to more ambiguity for some of the IGV-assessed candidate reads sites. Low read depth at a candidate site makes it hard for any software program or visual assessment to distinguish between a true variation and sequencing errors. The trios sequenced more recently where the average read depth was higher (e.g. 35) had more accurate identification by both software packages. The higher read depth provided a platform that was easier for both PedFilter and DenovoGear to pick out which reads were likely sequencing errors and which were true variations and therefore potentially DNMs.

In many human disease research studies, 'high coverage' WGS starts at a read depth of 20, but for most studies, a minimum read depth of 30 is recommended for more accurate results, with even higher read depths gaining higher sensitivity and accuracy (Gonzaga-Jauregui et al., 2012; Sun et al., 2021; Xu et al., 2017). The sequencing company Illumina recommends a read depth of 30 to 50 for WGS (*Sequencing Coverage for NGS Experiments*, n.d.). For the samples in this study, the average read depth was closer to 20 for most of the animals. While this does meet the minimum average read depth requirement, it does not allow for the most accurate analysis. This

was most noticeable in the candidates that were identified by both software packages. Most of the false positive or ambiguous candidates identified by both PedFilter and DenovoGear were for candidate DNMs that had a low read depth (less than 20). However when considering read depth, the higher the depth the more costly it is to sequence which can reduce the number of animals that can be sequenced within the research budget. For LIC, animals are sequenced with a target of 20-25 read depth to trade off the cost with the ability to sequence more individuals and potentially identify new variants or DNMs of interest. This is why a read depth of 20 was decided and used for this study.

Another feature specific to PedFilter that helped the identification of candidates when read depth was low is the use of population data. Unlike DenovoGear, which investigates trios individually, PedFilter uses information from the trio as well as from population data to see if this variation already exists in unrelated individuals. By taking this information into account, PedFilter is better able to judge if a variant is a valid DNM candidate. This is particularly useful when read depth is low and sequencing errors are more likely to be misread and flagged as potential DNM candidates.

Along with these differences, PedFilter and DenovoGear do have some similarities. While they put different emphases on precision and sensitivity, they both use variant data generated by other software programs and then use Bayesian statistical modelling to calculate the probability of a variant being a DNM. They are both also computationally heavy. Both PedFilter and DenovoGear deal with large variant file inputs that require large amounts of RAM and time to complete the statistical analyses and probability calculations. Breaking the input files into smaller amounts of data can speed up the process and free up memory to run other jobs/programs. To do this though requires specialised pipelines to be built with an understanding of how the variant files store and group information, what the expected output data should look like and how the pieces will fit back together if needed.

Due to the complexity and computing power required to run these DNM software packages, most pipelines and commands need to be written and built individually for each project. As outlined previously, each DNM software package used here was built for a specific single project and made open source with little further development. This has led to many DNM software packages being developed with one task in mind under very specific conditions. Due to this

specificity, the usability of these programs is very limited, with a lot of reconfiguration needed to make these programs usable for other or secondary research projects. As these projects are not professionally maintained, it is also difficult to get support or help related to the software packages. This low production and maintenance quality often means there are more benefits to creating a new specialised DNM identification software package over attempting to alter an existing package to meet the requirements of a new experiment.

The study of DNMs and the role they play in human and animal health and disease is still a new and rapidly growing research discipline. Because of this, it is expected that the research methods and software packages being used are still developing. The problem with continually creating new DNM software packages is that the quality and usability of these programs do not improve. It also means that common or accepted standards of practice are not likely to be established, making study comparisons and reviews difficult. The broad development of new software packages to improve our current ability to identify DNMs is needed but should occur alongside the deeper development of current packages. In doing this we can start to build a library of comparable studies as well as have one or two regularly maintained and updated point-of-reference software packages, to compare against newer software methods.

Pros and cons of using PedFilter and DenovoGear Software Packages

The purpose of this study was to directly compare DNM software packages and how well they worked. In the process of testing and running PedFilter and DenovoGear, there were some obvious benefits and struggles with both packages.

For PedFilter some of the benefits were related to usability: compared to DenovoGear, PedFilter was easier to set up and run. The input data for PedFilter was sectioned out by chromosome making for a reduced run time, and the output data was very extensive with additional information including trinucleotide, the probability it could be a mendelian instead of DNM, frequency of SNP in the submitted population among other things. Because of this, PedFilter output data can be easily used for deeper extended research and if more or different information is needed it can quickly be rerun for more data.

Two other benefits of PedFilter that should be noted, but were not taken advantage of for this project, were its ability to work with incomplete multi-generational pedigrees, and the use of

population variant sets to accommodate lower-quality cheaper sequence data. Because of the trio pedigree requirements of DenovoGear, the only research trios used were fully complete single-generation trios so the multi-generational aspect of PedFilter was not utilised. Population data is a requirement for PedFilter and is used to see if potential DNM candidates from a trio already exist in the population. This is particularly useful when working with large databases of potentially incomplete familial trios, or low-quality sequences as it adds more information that the algorithm can use to more accurately identify DNM candidates.

Alongside all these benefits there are a few downsides to using PedFilter. The biggest of these is the need for population data. The population data helps PedFilter identify candidate DNMs but it does mean that smaller or specified studies with a limited number of sequenced individuals will be unable to run this software package. The other con for PedFilter is that it has been designed for Bovine genetic sequences. Without some preliminary testing, this could affect how well it works on human or other organisms' genetic sequences.

DenovoGear also poses its set of unique pros and cons. DenovoGear is one of the older DNM-identifying software packages and because of this has been used in multiple studies since it was released in 2013. This has allowed time for DenovoGear to undergo multiple rounds of testing and updating compared to newer DNM software packages. One of these updates is a plugin to Bcftools which allows a DNG tag to be added to bcf/vcf files for each genetic variation noted within the files. This plugin makes DenovoGear user-friendly and easily accessible to those scientists or analysts already using Samtools and Bcftools. DenovoGear, like most current DNM software packages, works with complete single trios. This means that DenovoGear is capable of being used for smaller specified studies as well as larger, more robust ones.

With these advantages of using DenovoGear, there are also some drawbacks. When using the plugin with Bcftools, every variant gets a DNG probability and is included in a file with exponentially more non-DNM SNPs that are not relevant to the research. This then requires post-run filtering to extract relevant information as well as exclude highly unlikely candidates. Using the standalone software package would likely require less post-run processing; however, it is very time-consuming and computationally heavy while still requiring the creation of the bcf/vcf file. All of this while not being as easy to run as the plugin.

Lastly, being one of the first of its kind, DenovoGear has been used as the accuracy measuring stick for most DNM software packages that followed it. While inherently not a bad thing, most of these programs have built on from DenovoGear and claim a higher DNM identification accuracy. Most studies that conducted research with DenovoGear used it as a primary test with the output then being used in some other filtering, program or manual confirmation technique (C Yuen et al., 2017; Francioli et al., 2017; Tang et al., 2018; Wright et al., 2018). Together this suggests that, while DenovoGear provides a good starting point for narrowing down potential DNM candidates, it is not accurate enough to be used with complete authority.

Recommendations

Based on the data from this study regarding PedFilter and DenovoGear, the following four recommendations are made. For smaller studies (e.g., 3–15 trios), with limited data available to them, using DenovoGear with a secondary software program for filtering would be best. This is because the small amount of data would be fast to run and the output would also be quick to manually (or computationally) validate. This could be for studies investigating a small specific set of trios that are trying to capture all true positives, such as rare human genetic diseases like Schinzel-Giedion syndrome, Kabuki syndrome and Bohring-Opitz syndrome (Bögershausen & Wollnik, 2013; Hoischen et al., 2010, 2011; Veltman & Brunner, 2012). These types of studies often use high-quality sequences on a low number of affected individuals and, due to the low sample numbers, it is practical to perform manual validation.

If a study has sufficient samples to provide a simulated population data set, PedFilter has higher accuracy than DenovoGear on its own. PedFilter works well with both lower and higher-quality data provided there is adequate population data to support the analysis of the algorithm. This would work best for large long-term studies tracking genetic shifts or gains within a population, such as large animal breeding programs; this was the original purpose of PedFilter.

The best option for fast analysis without requiring heavy filtering of the output data would be a combination of both PedFilter and DenovoGear. The cross-over data from these two software packages gave the highest true positive and mosaic percentages. While this did miss a large number of true positives that were found only by one of the software packages, for a preliminary or additional test, the combination provides the quickest, most accurate idea of what DNMs are

present. This would be best for a study with a low budget and low-quality data, or a way to add more information to a broader genetics study.

Overall, neither software package is 100% accurate, and output data should be used only after some form of secondary filtering or validation. As has been noted in numerous DNM research papers, there are no set methods yet, and different algorithmic and filtering techniques do affect the output data and findings (Bergeron et al., 2022; De Summa et al., 2017; Lefouili & Nam, 2022; Liu et al., 2022; Ségurel et al., 2014). Current software selection and use are based on the differing priorities of each study.

In conclusion, more research is needed to form a better understanding of DNMs and their detection. Further research could be a new paper where more DNM candidates would undergo visual assessment using IGV or similar tools. Along with this, wet lab assessment of all potential mosaics, ambiguous and true positive candidates that were passed by visual inspection should be verified through targeted high-quality resequencing. This information would help strengthen or correct the findings made by any future paper.

A deeper study could include testing more software packages, such as testing DenovoCNN and Rufus or branching out to other packages like SynthDNM (Lian et al., 2021), DNMFilter_INDELs (Liu et al., 2020) or HAPDeNovo (Zhou et al., 2018). This would build a portfolio of direct comparisons that could help outline more promising areas of development and provide useful recommendations for future studies to understand and select from.

References

Acuna-Hidalgo, R., Veltman, J. A., & Hoischen, A. (2016). New insights into the generation and role of de novo mutations in health and disease. *Genome Biology*, *17*(1), 241. https://doi.org/10.1186/s13059-016-1110-1

Alexandrov, L. B., Nik-Zainal, S., Wedge, D. C., Aparicio, S. A. J. R., Behjati, S., Biankin, A. V., Bignell, G. R., Bolli, N., Borg, A., Børresen-Dale, A.-L., Boyault, S., Burkhardt, B., Butler, A. P., Caldas, C., Davies, H. R., Desmedt, C., Eils, R., Eyfjörd, J. E., Foekens, J. A., … Stratton, M. R. (2013). Signatures of mutational processes in human cancer. *Nature*, *500*(7463), Article 7463. https://doi.org/10.1038/nature12477

Bergeron, L. A., Besenbacher, S., Turner, T., Versoza, C. J., Wang, R. J., Price, A. L., Armstrong, E., Riera, M., Carlson, J., Chen, H., Hahn, M. W., Harris, K., Kleppe, A. S., López-Nandam, E. H., Moorjani, P., Pfeifer, S. P., Tiley, G. P., Yoder, A. D., Zhang, G., & Schierup, M. H. (2022). The Mutationathon highlights the importance of reaching standardization in estimates of pedigree-based germline mutation rates. *ELife*, *11*, e73577. https://doi.org/10.7554/eLife.73577

Bögershausen, N., & Wollnik, B. (2013). Unmasking Kabuki syndrome. *Clinical Genetics*, *83*(3), 201–211. https://doi.org/10.1111/cge.12051

C Yuen, R. K., Merico, D., Bookman, M., L Howe, J., Thiruvahindrapuram, B., Patel, R. V., Whitney, J., Deflaux, N., Bingham, J., Wang, Z., Pellecchia, G., Buchanan, J. A., Walker, S., Marshall, C. R., Uddin, M., Zarrei, M., Deneault, E., D'Abate, L., Chan, A. J. S., … Scherer, S. W. (2017). Whole genome sequencing resource identifies 18 new candidate genes for autism spectrum disorder. *Nature Neuroscience*, *20*(4), Article 4. https://doi.org/10.1038/nn.4524

Chen, T. (2019). All Versus One: An Empirical Comparison on Retrained and Incremental Machine Learning for Modeling Performance of Adaptable Software. *2019 IEEE/ACM 14th International*

Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 157–168. https://doi.org/10.1109/SEAMS.2019.00029

Crow, J. F. (2000). The origins, patterns and implications of human spontaneous mutation. *Nature Reviews Genetics*, *1*(1), Article 1. https://doi.org/10.1038/35049558

Danecek, P., Bonfield, J. K., Liddle, J., Marshall, J., Ohan, V., Pollard, M. O., Whitwham, A., Keane, T., McCarthy, S. A., Davies, R. M., & Li, H. (2021). Twelve years of SAMtools and BCFtools. *GigaScience*, *10*(2), giab008. https://doi.org/10.1093/gigascience/giab008

Davis, S. R., Spelman, R. J., & Littlejohn, M. D. (2017). BREEDING AND GENETICS SYMPOSIUM:Breeding heat tolerant dairy cattle: The case for introgression of the "slick" prolactin receptor variant into Bos taurus dairy breeds1. *Journal of Animal Science*, *95*(4), 1788–1800. https://doi.org/10.2527/jas.2016.0956

De Summa, S., Malerba, G., Pinto, R., Mori, A., Mijatovic, V., & Tommasi, S. (2017). GATK hard filtering: Tunable parameters to improve variant calling for next generation sequencing targeted gene panel data. *BMC Bioinformatics*, *18*(5), 119. https://doi.org/10.1186/s12859-017-1537-8

Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107–113. https://doi.org/10.1145/1327452.1327492

DePristo, M. A., Banks, E., Poplin, R., Garimella, K. V., Maguire, J. R., Hartl, C., Philippakis, A. A., del Angel, G., Rivas, M. A., Hanna, M., McKenna, A., Fennell, T. J., Kernytsky, A. M., Sivachenko, A. Y., Cibulskis, K., Gabriel, S. B., Altshuler, D., & Daly, M. J. (2011). A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature Genetics*, *43*(5), Article 5. https://doi.org/10.1038/ng.806

do Valle, Í. F., Giampieri, E., Simonetti, G., Padella, A., Manfrini, M., Ferrari, A., Papayannidis, C., Zironi, I., Garonzi, M., Bernardi, S., Delledonne, M., Martinelli, G., Remondini, D., &

Castellani, G. (2016). Optimized pipeline of MuTect and GATK tools to improve the detection of somatic single nucleotide polymorphisms in whole-exome sequencing data. *BMC Bioinformatics*, *17*(12), 341. https://doi.org/10.1186/s12859-016-1190-7

Drake, J. W., Charlesworth, B., Charlesworth, D., & Crow, J. F. (1998). Rates of Spontaneous Mutation. *Genetics*, *148*(4), 1667–1686. https://doi.org/10.1093/genetics/148.4.1667

El Naqa, I., & Murphy, M. J. (2015). What Is Machine Learning? In I. El Naqa, R. Li, & M. J. Murphy (Eds.), *Machine Learning in Radiation Oncology: Theory and Applications* (pp. 3–11). Springer International Publishing. https://doi.org/10.1007/978-3-319-18305-3_1

Elmeleegy, K., Sears, R., Condie, T., Conway, N., Alvaro, P., & . Hellerstein, J. M. (2010). *MapReduce Online*. 15.

Farrell, A. R. (2014). *Expanding the horizons of next generation sequencing with RUFUS* [Boston College]. http://dlib.bc.edu/islandora/object/bc-ir:104176

Farrell, J. A. R. (2022). *RUFUS* [C++]. https://github.com/jandrewrfarrell/RUFUS (Original work published 2014)

Francioli, L. C., Cretu-Stancu, M., Garimella, K. V., Fromer, M., Kloosterman, W. P., Samocha, K. E., Neale, B. M., Daly, M. J., Banks, E., DePristo, M. A., & de Bakker, P. I. (2017). A framework for the detection of de novo mutations in family-based sequencing data. *European Journal of Human Genetics*, *25*(2), Article 2. https://doi.org/10.1038/ejhg.2016.147

Fryxell, K. J., & Moon, W.-J. (2005). CpG mutation rates in the human genome are highly dependent on local GC content. *Molecular Biology and Evolution*, *22*(3), 650–658. https://doi.org/10.1093/molbev/msi043

Genome Research Limited. (2022, August 18). *Bgzip(1) manual page*. http://www.htslib.org/doc/bgzip.html

Gill, S. K., Christopher, A. F., Gupta, V., & Bansal, P. (2016). Emerging role of bioinformatics tools and software in evolution of clinical research. *Perspectives in Clinical Research*, *7*(3), 115–122. https://doi.org/10.4103/2229-3485.184782

Goldmann, J. M., Seplyarskiy, V. B., Wong, W. S. W., Vilboux, T., Neerincx, P. B., Bodian, D. L., Solomon, B. D., Veltman, J. A., Deeken, J. F., Gilissen, C., & Niederhuber, J. E. (2018). Germline de novo mutation clusters arise during oocyte aging in genomic regions with high double-strand-break incidence. *Nature Genetics*, *50*(4), Article 4. https://doi.org/10.1038/s41588-018-0071-6

Goldmann, J. M., Veltman, J. A., & Gilissen, C. (2019). De Novo Mutations Reflect Development and Aging of the Human Germline. *Trends in Genetics*, *35*(11), 828–839. https://doi.org/10.1016/j.tig.2019.08.005

Gonzaga-Jauregui, C., Lupski, J. R., & Gibbs, R. A. (2012). Human Genome Sequencing in Health and Disease. *Annual Review of Medicine*, *63*, 35–61. https://doi.org/10.1146/annurev-med-051010-162644

Haldane, J. B. S. (1935). The rate of spontaneous mutation of a human gene. *Journal of Genetics*, *83*(3), 235–244. https://doi.org/10.1007/BF02717892

Harland, C., Charlier, C., Karim, L., Cambisano, N., Deckers, M., Mni, M., Mullaart, E., Coppieters, W., & Georges, M. (2017). *Frequency of mosaicism points towards mutation-prone early cleavage cell divisions in cattle* (p. 079863). bioRxiv. https://doi.org/10.1101/079863

Harland, C., Mullaart, E., & Durkin, k. (2018). *Rate of de novo mutation in dairy cattle and potential impact of reproductive technologies*.

Harris, B. (2005). Breeding dairy cows for the future in New Zealand. *New Zealand Veterinary Journal*, *53*(6), 384–389. https://doi.org/10.1080/00480169.2005.36582

Hayes, B. J., & Daetwyler, H. D. (2019). 1000 Bull Genomes Project to Map Simple and Complex Genetic Traits in Cattle: Applications and Outcomes. *Annual Review of Animal Biosciences*, *7*(1), 89–102. https://doi.org/10.1146/annurev-animal-020518-115024

Heller, C. G., & Clermont, Y. (1963). Spermatogenesis in Man: An Estimate of Its Duration. *Science*, *140*(3563), 184–186. https://doi.org/10.1126/science.140.3563.184

Hoischen, A., van Bon, B. W. M., Gilissen, C., Arts, P., van Lier, B., Steehouwer, M., de Vries, P., de Reuver, R., Wieskamp, N., Mortier, G., Devriendt, K., Amorim, M. Z., Revencu, N., Kidd, A., Barbosa, M., Turner, A., Smith, J., Oley, C., Henderson, A., … Veltman, J. A. (2010). De novo mutations of SETBP1 cause Schinzel-Giedion syndrome. *Nature Genetics*, *42*(6), Article 6. https://doi.org/10.1038/ng.581

Hoischen, A., van Bon, B. W. M., Rodríguez-Santiago, B., Gilissen, C., Vissers, L. E. L. M., de Vries, P., Janssen, I., van Lier, B., Hastings, R., Smithson, S. F., Newbury-Ecob, R., Kjaergaard, S., Goodship, J., McGowan, R., Bartholdi, D., Rauch, A., Peippo, M., Cobben, J. M., Wieczorek, D., … de Vries, B. B. B. A. (2011). De novo nonsense mutations in ASXL1 cause Bohring-Opitz syndrome. *Nature Genetics*, *43*(8), Article 8. https://doi.org/10.1038/ng.868

Jay, M. (2007). The political economy of a productivist agriculture: New Zealand dairy discourses. *Food Policy*, *32*(2), 266–279. https://doi.org/10.1016/j.foodpol.2006.09.002

Kavikondala, A., Muppalla, V., Prakasha, Dr. K., & Acharya, V. (2019). *Automated Retraining of Machine Learning Models*. *8*, 445–452. https://doi.org/10.35940/ijitee.L3322.1081219

Kolesnikov, A., Goel, S., Nattestad, M., Yun, T., Baid, G., Yang, H., McLean, C. Y., Chang, P.-C., & Carroll, A. (2021). *DeepTrio: Variant Calling in Families Using Deep Learning* (p. 2021.04.05.438434). bioRxiv. https://doi.org/10.1101/2021.04.05.438434

Kong, A., Frigge, M. L., Masson, G., Besenbacher, S., Sulem, P., Magnusson, G., Gudjonsson, S. A., Sigurdsson, A., Jonasdottir, A., Jonasdottir, A., Wong, W. S. W., Sigurdsson, G., Walters, G. B., Steinberg, S., Helgason, H., Thorleifsson, G., Gudbjartsson, D. F., Helgason, A., Magnusson, O. T., … Stefansson, K. (2012). Rate of de novo mutations and the importance of father's age to disease risk. *Nature*, *488*(7412), Article 7412. https://doi.org/10.1038/nature11396

Larrañaga, P., Calvo, B., Santana, R., Bielza, C., Galdiano, J., Inza, I., Lozano, J. A., Armañanzas, R., Santafé, G., Pérez, A., & Robles, V. (2006). Machine learning in bioinformatics. *Briefings in Bioinformatics*, *7*(1), 86–112. https://doi.org/10.1093/bib/bbk007

Lefouili, M., & Nam, K. (2022). The evaluation of Bcftools mpileup and GATK HaplotypeCaller for variant calling in non-human species. *Scientific Reports*, *12*(1), Article 1. https://doi.org/10.1038/s41598-022-15563-2

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., & 1000 Genome Project Data Processing Subgroup. (2009). The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, *25*(16), 2078–2079. https://doi.org/10.1093/bioinformatics/btp352

Lian, A., Guevara, J., Xia, K., & Sebat, J. (2021). Customized de novo mutation detection for any variant calling pipeline: SynthDNM. *Bioinformatics*, *37*(20), 3640–3641. https://doi.org/10.1093/bioinformatics/btab225

Littlejohn, M. D., Henty, K. M., Tiplady, K., Johnson, T., Harland, C., Lopdell, T., Sherlock, R. G., Li, W., Lukefahr, S. D., Shanks, B. C., Garrick, D. J., Snell, R. G., Spelman, R. J., & Davis, S. R. (2014). Functionally reciprocal mutations of the prolactin signalling pathway define hairy and slick cattle. *Nature Communications*, *5*(1), Article 1. https://doi.org/10.1038/ncomms6861

Liu, J., Shen, Q., & Bao, H. (2022). Comparison of seven SNP calling pipelines for the next-generation sequencing data of chickens. *PLOS ONE*, *17*(1), e0262574. https://doi.org/10.1371/journal.pone.0262574

Liu, Y., Liu, J., & Wang, Y. (2020). Filtering de novo indels in parent-offspring trios. *BMC Bioinformatics*, *21*(16), 547. https://doi.org/10.1186/s12859-020-03900-z

McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., Garimella, K., Altshuler, D., Gabriel, S., Daly, M., & DePristo, M. A. (2010). The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Research*, *20*(9), 1297–1303. https://doi.org/10.1101/gr.107524.110

Men, A. E., Wilson, P., Siemering, K., & Forrest, S. (2008). Sanger DNA Sequencing. In *Next Generation Genome Sequencing* (pp. 1–11). John Wiley & Sons, Ltd. https://doi.org/10.1002/9783527625130.ch1

Mohiuddin, M., Kooy, R. F., & Pearson, C. E. (2022). De novo mutations, genetic mosaicism and human disease. *Frontiers in Genetics*, *13*. https://www.frontiersin.org/articles/10.3389/fgene.2022.983668

Nik-Zainal, S., Alexandrov, L. B., Wedge, D. C., Van Loo, P., Greenman, C. D., Raine, K., Jones, D., Hinton, J., Marshall, J., Stebbings, L. A., Menzies, A., Martin, S., Leung, K., Chen, L., Leroy, C., Ramakrishna, M., Rance, R., Lau, K. W., Mudie, L. J., … Stratton, M. R. (2012). Mutational Processes Molding the Genomes of 21 Breast Cancers. *Cell*, *149*(5), 979–993. https://doi.org/10.1016/j.cell.2012.04.024

Ostrander, B. E. P., Butterfield, R. J., Pedersen, B. S., Farrell, A. J., Layer, R. M., Ward, A., Miller, C., DiSera, T., Filloux, F. M., Candee, M. S., Newcomb, T., Bonkowsky, J. L., Marth, G. T., & Quinlan, A. R. (2018). Whole-genome analysis for effective clinical diagnosis and gene

discovery in early infantile epileptic encephalopathy. *Npj Genomic Medicine*, *3*(1), Article 1. https://doi.org/10.1038/s41525-018-0061-8

Ramu, A., Noordam, M. J., Schwartz, R. S., Wuster, A., Hurles, M. E., Cartwright, R. A., & Conrad, D. F. (2013). DeNovoGear: De novo indel and point mutation discovery and phasing. *Nature Methods*, *10*(10), Article 10. https://doi.org/10.1038/nmeth.2611

Rice, E. S., & Green, R. E. (2019). New Approaches for Genome Assembly and Scaffolding. *Annual Review of Animal Biosciences*, *7*(1), 17–40. https://doi.org/10.1146/annurev-animal-020518-115344

Robinson, J. T., Thorvaldsdóttir, H., Winckler, W., Guttman, M., Lander, E. S., Getz, G., & Mesirov, J. P. (2011). Integrative genomics viewer. *Nature Biotechnology*, *29*(1), Article 1. https://doi.org/10.1038/nbt.1754

Sawicki, M. P., Samara, G., Hurwitz, M., & Passaro, E. (1993). Human Genome Project. *The American Journal of Surgery*, *165*(2), 258–264. https://doi.org/10.1016/S0002-9610(05)80522-7

Ségurel, L., Wyman, M. J., & Przeworski, M. (2014). Determinants of Mutation Rate Variation in the Human Germline. *Annual Review of Genomics and Human Genetics*, *15*(1), 47–70. https://doi.org/10.1146/annurev-genom-031714-125740

Sharma, S., Wistuba, J., Pock, T., Schlatt, S., & Neuhaus, N. (2019). Spermatogonial stem cells: Updates from specification to clinical relevance. *Human Reproduction Update*, *25*(3), 275–297. https://doi.org/10.1093/humupd/dmz006

Sun, Y., Liu, F., Fan, C., Wang, Y., Song, L., Fang, Z., Han, R., Wang, Z., Wang, X., Yang, Z., Xu, Z., Peng, J., Shi, C., Zhang, H., Dong, W., Huang, H., Li, Y., Le, Y., Sun, J., & Peng, Z. (2021). Characterizing sensitivity and coverage of clinical WGS as a diagnostic test for genetic disorders. *BMC Medical Genomics*, *14*(1), 102. https://doi.org/10.1186/s12920-021-00948-5

Tang, C. S., Zhuang, X., Lam, W.-Y., Ngan, E. S.-W., Hsu, J. S., Michelle, Y. U., Man-Ting, S. O., Cherny, S. S., Ngo, N. D., Sham, P. C., Tam, P. K., & Garcia-Barcelo, M.-M. (2018). Uncovering the genetic lesions underlying the most severe form of Hirschsprung disease by whole-genome sequencing. *European Journal of Human Genetics*, *26*(6), Article 6. https://doi.org/10.1038/s41431-018-0129-z

Trapnell, C., & Salzberg, S. L. (2009). How to map billions of short reads onto genomes. *Nature Biotechnology*, *27*(5), Article 5. https://doi.org/10.1038/nbt0509-455

Veltman, J. A., & Brunner, H. G. (2012). De novo mutations in human genetic disease. *Nature Reviews Genetics*, *13*(8), Article 8. https://doi.org/10.1038/nrg3241

Venn, O., Turner, I., Mathieson, I., de Groot, N., Bontrop, R., & McVean, G. (2014). Strong male bias drives germline mutation in chimpanzees. *Science*, *344*(6189), 1272–1275. https://doi.org/10.1126/science.344.6189.1272

Wilkinson, D. J. (2007). Bayesian methods in bioinformatics and computational systems biology. *Briefings in Bioinformatics*, *8*(2), 109–116. https://doi.org/10.1093/bib/bbm007

Wright, C. F., McRae, J. F., Clayton, S., Gallone, G., Aitken, S., FitzGerald, T. W., Jones, P., Prigmore, E., Rajan, D., Lord, J., Sifrim, A., Kelsell, R., Parker, M. J., Barrett, J. C., Hurles, M. E., FitzPatrick, D. R., & Firth, H. V. (2018). Making new genetic diagnoses with old data: Iterative reanalysis and reporting from genome-wide data in 1,133 families with developmental disorders. *Genetics in Medicine*, *20*(10), 1216–1223. https://doi.org/10.1038/gim.2017.246

Xu, C., Wu, K., Zhang, J., Shen, H., & Deng, H.-W. (2017). Low-, high-coverage and two-stage DNA sequencing in the design of the genetic association study. *Genetic Epidemiology*, *41*(3), 187–197. https://doi.org/10.1002/gepi.22015

Yochem, J., & Herman, R. K. (2005). Genetic mosaics. In *WormBook: The Online Review of C. elegans Biology [Internet]*. WormBook. https://www.ncbi.nlm.nih.gov/books/NBK19725/

Zhang, Y., & Rajapakse, J. C. (2009). *Machine Learning in Bioinformatics*. John Wiley & Sons.

Zhou, X., Batzoglou, S., Sidow, A., & Zhang, L. (2018). HAPDeNovo: A haplotype-based approach for filtering and phasing de novo mutations in linked read sequencing data. *BMC Genomics*, *19*(1), 467. https://doi.org/10.1186/s12864-018-4867-7

Zhou, Y., He, F., Pu, W., Gu, X., Wang, J., & Su, Z. (2020). The Impact of DNA Methylation Dynamics on the Mutation Rate During Human Germline Development. *G3: Genes|Genomes|Genetics*, *10*(9), 3337–3346. https://doi.org/10.1534/g3.120.401511

Base Quality Score Recalibration (BQSR). (n.d.). GATK. Retrieved 13 June 2022, from https://gatk.broadinstitute.org/hc/en-us/articles/360035890531-Base-Quality-Score-Recalibration-BQSR-

BaseRecalibrator. (n.d.). GATK. Retrieved 13 June 2022, from https://gatk.broadinstitute.org/hc/en-us/articles/360036898312-BaseRecalibrator

COSMIC | SBS - Mutational Signatures. (n.d.). Retrieved 29 March 2023, from https://cancer.sanger.ac.uk/signatures/sbs/

Creating a NeSI Account Profile. (2022). NeSI Support. https://support.nesi.org.nz/hc/en-gb/articles/360000159715-Creating-a-NeSI-Account-Profile

DeNovoGear—Estimating de novo mutations from related individuals and cells. (2022). [C++]. Ultimate Source Code Project. https://github.com/ultimatesource/denovogear (Original work published 2013)

Intro to data structures—Pandas 1.5.2 documentation. (2022). https://pandas.pydata.org/docs/user_guide/dsintro.html

Jupyter on NeSI. (2022, August 12). NeSI Support. https://support.nesi.org.nz/hc/en-gb/articles/360001555615-Jupyter-on-NeSI

New Zealand eScience Infrastructure. (2019). New Zealand EScience Infrastructure. https://www.nesi.org.nz/

Releases · broadinstitute/gatk. (n.d.). GitHub. Retrieved 7 December 2022, from https://github.com/broadinstitute/gatk/releases

Sequencing Coverage for NGS Experiments. (n.d.). Retrieved 21 March 2023, from https://www.illumina.com/science/technology/next-generation-sequencing/plan-experiments/coverage.html

Tuakiri—Trust and Identity: REANNZ. (2022). https://www.reannz.co.nz/products-and-services/tuakiri/join/

Appendices

Appendix A: Scripts and Commands

A.i: DenovoGear query command

```
####################BETTER COMMAND INCLUDES MORE DATA AND HEADER
bcftools query -H -S /nesi/project/nesi00187/erdue0/Trios_for_Analysis/crams/sampleIDList.txt -i '(DNM != ".") & (MQ > 50)'
-f '[%CHROM\t%POS\t%SAMPLE\t%REF\t%ALT\t%GT\t%DNM\n]' /nesi/project/nesi00187/ARS-UCD1.2/vcfs/2022-03-
22_BCFTOOLS_1800_animals/BCFTools_March_2022_1830anmls.bcf -o DenovoGear_samplesOnly.txt &
```

A. i: PedFilter Average read-depth script

```
#!/bin/bash
#SBATCH -c 1 --mem=256M --time 1-00 -J 12573
#SBATCH --account=nesi00187     # Project Account
#set up above for current nesi project

CHROM=$SLURM_ARRAY_TASK_ID #for counting when using an array
echo ${CHROM}
echo "grep "^PSC" PedFilterStats/PSCstatsChr"${CHROM}".stats |awk 'OFS="\t" {print $3,$10}' > avgDepth_"${CHROM}".stats"
grep "^PSC" PedFilterStats/PSCstatsChr${CHROM}.stats |awk 'OFS="\t" {print $3,$10}' > avgDepth_${CHROM}.stats
```

A. i: PedFilter Bulk Run

```
#!/bin/bash
#SBATCH -c 16 --mem=78G --account=nesi00187 --time 7-00 -J 12573
### Ask for ~16 CPU and 1.3x -J-Xmx60g ram.
##This extra part of RAM is for the java virtual machine required to run the program NOTE:do this for all java based programs

#this is the WORKING run script for the selected trio with extra population sample chrom vcf files



module load GATK
CHROM=$SLURM_ARRAY_TASK_ID
echo ${CHORM}
echo /nesi/nobackup/nesi00187/erdue0/PedFilterChrFiles/gatk.march-2022.refine.chr${CHROM}_erdue0.vcf.gz
echo /nesi/nobackup/nesi00187/erdue0/PedFilterStats/avgDepth_${CHROM}.txt

./scala-2.12.14/bin/scala -J-Xmx60g -cp *.jar:. pedFilter.pedigreeFilter VCF=/nesi/nobackup/nesi00187/erdue0/PedFilterChrFiles/gatk.march-
2022.refine.chr${CHROM}_erdue0.vcf.gz PED=/nesi/project/nesi00187/erdue0/Trios_for_Analysis/trios.ped
TRIOS=/nesi/nobackup/nesi00187/erdue0/PedFilterStats/avgDepth_${CHROM}.txt phase=/nesi/project/nesi00187/ARS-
UCD1.2/vcfs/vqsr_filtering/GATK_AUTOZOMES_Combined_INDEL99.0_SNP99.95_PASS_ONLY.vcf.gz type=gatk ref=/nesi/project/nesi00187/indexed-
genomes/bos_taurus/ARS-UCD1.2_lic_less_alts/ARS_lic_less_alts.male.fa maxDPx=2.0 minDP=10 minALT=1 RECUR=F minKIDS=0 minPL=0 QUAL=0
minRAFQ=0.2 mapQ=60 -O /nesi/project/nesi00187/erdue0/PedFilterOutput_gatk.march-2022.refine.chr${CHROM}_erdue0.txt

#VCF=/nesi/nobackup/nesi00187/erdue0/PedFilterChrFiles/gatk.march-2022.refine.chr${CHROM}_erdue0.vcf.gz
#echo /nesi/nobackup/nesi00187/erdue0/PedFilterChrFiles/gatk.march-2022.refine.chr${CHROM}_erdue0.vcf.gz
#echo /nesi/nobackup/nesi00187/erdue0/PedFilterStats/avgDepth_${CHROM}.txt

#PHASE=/nesi/project/nesi00187/ARS-UCD1.2/vcfs/vqsr_filtering/GATK_AUTOZOMES_Combined_INDEL99.0_SNP99.95_PASS_ONLY.vcf.gz
#ped=/nesi/project/nesi00187/erdue0/Trios_for_Analysis/trios.ped
#trios=/nesi/nobackup/nesi00187/erdue0/PedFilterStats/avgDepth_${CHROM}.txt
#type=gatk
#ref=/nesi/project/nesi00187/indexed-genomes/bos_taurus/ARS-UCD1.2_lic_less_alts/ARS_lic_less_alts.male.fa
#minkids=0
```

A.iv DenovoCNN Base Recalibrator

```
#!/bin/bash
#SBATCH -c 4 --mem=24G  --account=nesi00187 --time 7-00 -J 12573

#this code is to be used as an array command, to outout recal table from bams to then be used for base quality score recalibration

module load SAMtools
module load GATK

echo modules loaded
TARGET=(`cat $1`)    #this is the first line in the provided file
echo ${TAREGET}
BAMFILE=${TARGET[$SLURM_ARRAY_TASK_ID]} #this is the correct line being used in the command
echo ${BAMFILE}

gatk BaseRecalibrator -I /nesi/project/nesi00187/erdue0/Trios_for_Analysis/crams/${BAMFILE} -R /home/ekd3/working/indexed-
genomes/bos_taurus/ARS-UCD1.2_lic_less_alts/ARS_lic_less_alts.male.fa --known-sites
/nesi/project/nesi00187/erdue0/GATK_AUTOZOMES_Combined_INDEL99.0_SNP99.95_PASS_ONLY.vcf.gz -O ${BAMFILE}_recal.table

echo gatk BaseRecalibrator -I /nesi/project/nesi00187/erdue0/Trios_for_Analysis/crams/${BAMFILE} -R /home/ekd3/working/indexed-
genomes/bos_taurus/ARS-UCD1.2_lic_less_alts/ARS_lic_less_alts.male.fa --known-sites
/nesi/project/nesi00187/erdue0/GATK_AUTOZOMES_Combined_INDEL99.0_SNP99.95_PASS_ONLY.vcf.gz -O ${BAMFILE}_recal.table
```

A.v DenovoCNN Base Quality Score Recalibration

```
#!/bin/bash
#SBATCH -c 8 --mem=24G  --account=nesi00187 --time 7-00 -J 12573
#order of files inputted, script.sh, BamList, recalList
module load SAMtools
module load GATK

TARGETBAM=(`cat $1`)
TARGETRECAL=(`cat $2`)
BAMFILE=${TARGETBAM[$SLURM_ARRAY_TASK_ID]} #this is the correct line being used in the command
RECALFILE=${TARGETRECAL[$SLURM_ARRAY_TASK_ID]}
echo ${RECALFILE}
echo ${BAMFILE}
gatk ApplyBQSR -R /home/ekd3/working/indexed-genomes/bos_taurus/ARS-UCD1.2_lic_less_alts/ARS_lic_less_alts.male.fa -I ${BAMFILE} --bqsr-
recal-file ${RECALFILE} -O BQSR_${BAMFILE}
echo gatk ApplyBQSR -R /home/ekd3/working/indexed-genomes/bos_taurus/ARS-UCD1.2_lic_less_alts/ARS_lic_less_alts.male.fa -I ${BAMFILE} --
bqsr-recal-file ${RECALFILE} -O BQSR_${BAMFILE}
```

A.vi: Samtools faidx script for Reference tri-nuc extraction for DenovoGear candidate DNMs

```
#!/bin/bash
#SBATCH -c 4 --mem=24G  --account=nesi00187 --time 7-00 -J 12573
#order of files inputted, script.sh, chrFileList.txt, posFileList.txt →(file needs to have column
chrom\tpos
module load SAMtools

declare -a POSARRAY
declare -a CHRARRAY
CHRARRAY=(`cat $1`)
POSARRAY=(`cat $2`)

echo ${POSARRAY[1]}
ARRAYLEN=${#POSARRAY[@]}

for ((i=1; i<${ARRAYLEN} ; i++))——→#start at 1 to miss the first header row
#do
{
——→START=$((POSARRAY[$i]-1))
——→END=$((POSARRAY[$i]+1))
——→TRINUC=${CHRARRAY[$i]}":"${START}"-"${END}

——→echo ${CHRARRAY[$i]}":"${START}"-"${END}
——→samtools faidx /nesi/project/nesi00187/indexed-genomes/bos_taurus/ARS-
UCD1.2_lic_less_alts/ARS_lic_less_alts.male.fa ${TRINUC} >> FinalFiles/dng_triNucRef.txt
}
```

Appendix B: Jupyter NoteBook for DenovoGear

[1]:
```
#testing import statements for whether nesi has installed them or not
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import csv
print ("Library imports successful")

#loading DNG data file into dataframe for processing and manipulation
denovogearDF = pd.read_csv("/nesi/project/nesi00187/erdue0/FinalFiles/
↪denovogear_noChrX_v2.csv", sep=",", header=0).astype(str)
# print(len(denovogearDF))
print(denovogearDF.head(5)) #print to see has loaded correctly

#loading pedFilter data file into separate dataframe for processing and␣
↪manipulation
print("#########################################################################################")
pedfilterDF = pd.read_csv("/nesi/project/nesi00187/erdue0/FinalFiles/
↪trioPedFilterCombine_v2.csv", sep=",", header=0).astype(str)
pedfilterDF.Genotype = pedfilterDF["Genotype"].str.replace("\'","")        #cleaning␣
↪input data of random unnecessary symbols
#removing unneccessary lines from where separate chrom files were appended into␣
↪1 file
pedfilterDF = pedfilterDF.loc[(pedfilterDF["Chrom"] != "nan") &␣
↪(pedfilterDF["Chrom"] != "Chrom") & (pedfilterDF["Chrom"] != "done") &␣
↪(pedfilterDF["Pos"] != "nan")]

# print(len(pedfilterDF))
print(pedfilterDF.head(5)) #print to see has loaded correctly

##############
##Error checking unwanted data was filtered out correctly from pedfilter and␣
↪denovogear dataframe
# Testing both dataframes for any genotypes that are not 0/1 (should be none)
print(denovogearDF.loc[denovogearDF["Genotype"] != "0/1"])
print(pedfilterDF.loc[pedfilterDF["Chrom"] == "Chrom"])

print("#####################################")
```

```
print(len(denovogearDF))

##############################
#Combining DNG and PEDfilter dataframes into sinlge entity. Outer join merges␣
 ↪data that was found in both packages and keeps the data that was only found␣
 ↪in 1
indexCombinedDF = pd.merge(pedfilterDF, denovogearDF,␣
↪on=["Chrom","Pos","SAMPLE",    "Ref"],    how="outer")
print(len(indexCombinedDF))
print(indexCombinedDF.head(5))
print(len(indexCombinedDF)) # to see that some data points have merged (dnm␣
 ↪candidates found by both software)
###################################
#Error checking combined dataframe before further processing
#checking there are no chrom data points that hold "Chrom" instead of chrom␣
 ↪number
print (indexCombinedDF.loc[indexCombinedDF["Chrom"] == "Chrom"])

# # used to test of any null variable points where there shouldn't be
#  indexCombinedDF = indexCombinedDF.loc[indexCombinedDF['Chrom'] != 'nan']
```

```
Library imports successful
  Unnamed: 0 Chrom      Pos Ref Alt SAMPLE Genotype      DNMsc
0          19  chr1   139169   A   C  trio5      0/1  -10.1314
1          23  chr1   139552   T   A  trio5      0/1  -18.4239
2          25  chr1   139553   C   A  trio5      0/1  -18.1666
3          27  chr1   139556   G   A  trio5      0/1  -17.8069
4          33  chr1   139564   T   C  trio5      0/1  -17.9226
################################################################################
##############################
  Unnamed: 0 Chrom      Pos rsID     TRI-NUC    Ref  Alt     QUAL SAMPLE  \
0          0  chr1   926385    .     CCA>CAA      C    A    11.62  trio3
1          1  chr1  3035478    .     ACA>AAA      C    A    112.8  trio2
2          2  chr1  3097802    .     TGG>TTG      G    T     24.6  trio3
3          3  chr1  3374744    .     AGA>AAA      G    A   375.13  trio2
4          4  chr1  3955781    .  AAG>INDEL AGAGG  A    316.07  trio1

  Genotype  ...  Proband              Sire  gSire  gDam              Dam  \
0      0/1  ...    (8,2)    (38,0)  0,99,1485   nan   nan   (21,0)  0,57,855
1      0/1  ...    (8,2)     (13,0)  0,0,251    nan   nan  (26,0)  0,66,1013
2      0/1  ...    (8,2)    (42,0)  0,103,1453  nan   nan   (14,0)  0,39,585
3      0/1  ...   (20,7)    (30,0)  0,81,1215   nan   nan   (32,0)  0,14,954
4      0/1  ...  (14,14)     (34,0)  0,57,855    nan   nan  (37,0)  0,72,1080

  gSire.1 gDam.1 PopAB         PhaseInfo Unnamed: 43
0     nan     nan  9232               0 0.0        nan
1     nan     nan  8572  10 0.0011652296           nan
```

```
2      nan     nan  8846              0 0.0          nan
3      nan     nan  9033    5 5.5321975E-4          nan
4      nan     nan  8826              0 0.0          nan

[5 rows x 45 columns]
Empty DataFrame
Columns: [Unnamed: 0, Chrom, Pos, Ref, Alt, SAMPLE, Genotype, DNMsc]
Index: []
Empty DataFrame
Columns: [Unnamed: 0, Chrom, Pos, rsID, TRI-NUC, Ref, Alt, QUAL, SAMPLE,
Genotype, PLs, Denovo-Posterior, Probs(mend,denovo,bad), MichelsCriteria,
Source, Phase, Denovo-S|D, Haps-S|D, Ances, Pars, Children, Desc, ExFam, Pop,
PopFreq, Support_Ratio, Offspring-Support-Ratio, Class, Proband_dif_kids,
Proband_dif_pop, Sire_Dif_Kids, Dam_dif_kids, _Sire_dif_pop, Dam_dif_Pop, PopAD,
Proband, Sire, gSire, gDam, Dam, gSire.1, gDam.1, PopAB, PhaseInfo, Unnamed: 43]
Index: []

[0 rows x 45 columns]
#######################################
947482
950455
  Unnamed: 0_x Chrom      Pos rsID     TRI-NUC    Ref Alt_x     QUAL SAMPLE \
0            0  chr1   926385    .     CCA>CAA      C     A    11.62  trio3
1            1  chr1  3035478    .     ACA>AAA      C     A    112.8  trio2
2            2  chr1  3097802    .     TGG>TTG      G     T     24.6  trio3
3            3  chr1  3374744    .     AGA>AAA      G     A   375.13  trio2
4            4  chr1  3955781    .  AAG>INDEL  AGAGG     A   316.07  trio1

  Genotype_x  ...          Dam gSire.1 gDam.1 PopAB          PhaseInfo  \
0        0/1  ...     (21,0)  0,57,855    nan    nan  9232              0 0.0
1        0/1  ...     (26,0)  0,66,1013   nan    nan  8572  10 0.0011652296
2        0/1  ...     (14,0)  0,39,585    nan    nan  8846              0 0.0
3        0/1  ...     (32,0)  0,14,954    nan    nan  9033    5 5.5321975E-4
4        0/1  ...     (37,0)  0,72,1080   nan    nan  8826              0 0.0

  Unnamed: 43 Unnamed: 0_y Alt_y Genotype_y DNMsc
0         nan          NaN   NaN        NaN   NaN
1         nan          NaN   NaN        NaN   NaN
2         nan          NaN   NaN        NaN   NaN
3         nan          NaN   NaN        NaN   NaN
4         nan          NaN   NaN        NaN   NaN

[5 rows x 49 columns]
950455
Empty DataFrame
Columns: [Unnamed: 0_x, Chrom, Pos, rsID, TRI-NUC, Ref, Alt_x, QUAL, SAMPLE,
Genotype_x, PLs, Denovo-Posterior, Probs(mend,denovo,bad), MichelsCriteria,
Source, Phase, Denovo-S|D, Haps-S|D, Ances, Pars, Children, Desc, ExFam, Pop,
```

PopFreq, Support_Ratio, Offspring-Support-Ratio, Class, Proband_dif_kids, Proband_dif_pop, Sire_Dif_Kids, Dam_dif_kids, _Sire_dif_pop, Dam_dif_Pop, PopAD, Proband, Sire, gSire, gDam, Dam, gSire.1, gDam.1, PopAB, PhaseInfo, Unnamed: 43, Unnamed: 0_y, Alt_y, Genotype_y, DNMsc]
Index: []

[0 rows x 49 columns]

[3]:
```
# # Code to add Column that tags each line as either being found by PedFiter, DenovoGear or Both

#add empty column to data frame where software id tag will be added
indexCombinedDF["software"]  =  np.nan

# header = header.reset_index()
# print (indexCombinedDF.head(5))
#  header  =  header.reset_index()
print (indexCombinedDF.head(5))

#Goes through each line and add a tag to the software column that identifies
for index, row in indexCombinedDF.iterrows():
    val = ''
    if (pd.notnull(row[11]) & pd.notnull(row["DNMsc"]) ):
        val = "both"
    elif (pd.notnull(row[11])):
        val = "ped"
    elif (pd.notnull(row["DNMsc"])):
        val = "dng"
    else:
        print("else entered")
        val = "error"
    indexCombinedDF.at[index, "software"] = val
print("for loop done (1/3)")

print(indexCombinedDF.head(5))  #making sure that some tags have been added correctly
print(indexCombinedDF.tail(5))
```

```
  Unnamed: 0_x Chrom      Pos rsID     TRI-NUC    Ref Alt_x     QUAL SAMPLE \
0            0  chr1   926385    .    CCA>CAA      C     A    11.62  trio3
1            1  chr1  3035478    .    ACA>AAA      C     A    112.8  trio2
2            2  chr1  3097802    .    TGG>TTG      G     T     24.6  trio3
3            3  chr1  3374744    .    AGA>AAA      G     A   375.13  trio2
4            4  chr1  3955781    .  AAG>INDEL  AGAGG     A   316.07  trio1

  Genotype_x  ... gSire.1 gDam.1 PopAB         PhaseInfo Unnamed: 43  \
0        0/1  ...     nan    nan  9232                0 0.0          nan
1        0/1  ...     nan    nan  8572  10 0.0011652296          nan
```

```
2         0/1  ...      nan     nan  8846              0 0.0          nan
3         0/1  ...      nan     nan  9033    5 5.5321975E-4          nan
4         0/1  ...      nan     nan  8826              0 0.0          nan

  Unnamed: 0_y Alt_y Genotype_y DNMsc software
0          NaN   NaN        NaN   NaN      NaN
1          NaN   NaN        NaN   NaN      NaN
2          NaN   NaN        NaN   NaN      NaN
3          NaN   NaN        NaN   NaN      NaN
4          NaN   NaN        NaN   NaN      NaN

[5 rows x 50 columns]
for loop done (1/3)
  Unnamed: 0_x Chrom      Pos rsID     TRI-NUC    Ref Alt_x     QUAL SAMPLE \
0            0  chr1   926385    .   CCA>CAA      C     A    11.62  trio3
1            1  chr1  3035478    .   ACA>AAA      C     A    112.8  trio2
2            2  chr1  3097802    .   TGG>TTG      G     T     24.6  trio3
3            3  chr1  3374744    .   AGA>AAA      G     A   375.13  trio2
4            4  chr1  3955781    .  AAG>INDEL  AGAGG     A   316.07  trio1

  Genotype_x  ... gSire.1 gDam.1 PopAB         PhaseInfo Unnamed: 43  \
0        0/1  ...      nan     nan  9232              0 0.0          nan
1        0/1  ...      nan     nan  8572   10 0.0011652296          nan
2        0/1  ...      nan     nan  8846              0 0.0          nan
3        0/1  ...      nan     nan  9033    5 5.5321975E-4          nan
4        0/1  ...      nan     nan  8826              0 0.0          nan

  Unnamed: 0_y Alt_y Genotype_y DNMsc software
0          NaN   NaN        NaN   NaN      ped
1          NaN   NaN        NaN   NaN      ped
2          NaN   NaN        NaN   NaN      ped
3          NaN   NaN        NaN   NaN      ped
4          NaN   NaN        NaN   NaN      ped

[5 rows x 50 columns]
        Unnamed: 0_x  Chrom       Pos rsID TRI-NUC  Ref Alt_x QUAL  SAMPLE  \
950450           NaN  chr29  51062394  NaN     NaN  CCC   NaN  NaN  trio5
950451           NaN  chr29  51070218  NaN     NaN    A   NaN  NaN  trio3
950452           NaN  chr29  51079711  NaN     NaN    T   NaN  NaN  trio3
950453           NaN  chr29  51081000  NaN     NaN    G   NaN  NaN  trio4
950454           NaN  chr29  51096068  NaN     NaN    A   NaN  NaN  trio2

        Genotype_x  ... gSire.1 gDam.1 PopAB PhaseInfo Unnamed: 43  \
950450         NaN  ...     NaN    NaN   NaN       NaN         NaN
950451         NaN  ...     NaN    NaN   NaN       NaN         NaN
950452         NaN  ...     NaN    NaN   NaN       NaN         NaN
950453         NaN  ...     NaN    NaN   NaN       NaN         NaN
950454         NaN  ...     NaN    NaN   NaN       NaN         NaN
```

```
        Unnamed: 0_y          Alt_y Genotype_y      DNMsc software
950450       5447033  CCTCC,CCCTCC        0/1  -18.8813      dng
950451       5447045              T        0/1  -20.2604      dng
950452       5447092              C        0/1  -14.5504      dng
950453       5447113              T        0/1   -17.037      dng
950454       5447148              G        0/1  -10.8551      dng

[5 rows x 50 columns]
```

[4]:
```
##Code for generating and assigning a tri-nucleotide (tri-nuc) to each␣
↪candidate
## formatted to match the automatically assigned tri-nuc for the ped filter data

dngtrinuc = indexCombinedDF.loc[(indexCombinedDF["software"] == "dng") |␣
↪(indexCombinedDF["software"] == "both")]   #subset DNG data into smaller␣
↪dataframe
dngtrinuc = dngtrinuc[["Chrom","Pos", "SAMPLE", "Ref", "Alt_y", "Genotype_y",␣
↪"DNMsc", "Denovo-Posterior", "software"]] #subsetting further to only␣
↪include relevant columns
dngtrinuc["DNMsc"]    =    pd.to_numeric(dngtrinuc["DNMsc"])
dngtrinuc["Denovo-Posterior"]  =  pd.to_numeric(dngtrinuc["Denovo-Posterior"])

#Removing and INDEL in reference data as can't make an accurate tri-nuc and␣
↪they are hard to identify conslusively as DNM
dngtrinuc = dngtrinuc.loc[(dngtrinuc.Ref == "A") | (dngtrinuc.Ref == "T") |␣
↪(dngtrinuc.Ref == "C") | (dngtrinuc.Ref == "G")]
# print(dngtrinuc.head(50))      #to test INDELs have been removed

#for loop for getting base represented by the 0/1 genotypes in this dataframe
 for index, row in dngtrinuc.iterrows():
   if (',' in dngtrinuc.Alt_y[index]):
        # print("TRUE")
        s    =    dngtrinuc.Alt_y[index].split(',')
        # print(s)
        dngtrinuc.Alt_y[index]  =  s[0]
        # print(dngtrinuc.Alt_y[index])


print ("DNG finished")
#removing any extra INDEL data from the alt data
#Alt removal needs to happen after for loop to account for the few SNP␣
↪genotypes that aren't 0/1
dngtrinuc = dngtrinuc.loc[(dngtrinuc.Alt_y == "A") | (dngtrinuc.Alt_y == "T") |␣
↪(dngtrinuc.Alt_y == "C") | (dngtrinuc.Alt_y == "G")]

##########################################
```

```
# following code only needs to be run once to create file. The produced files␣
 ↪will run through samtools

# dngtrinuc.to_csv("denovoGear_for_TriNuc.csv")

# dngtrinuc['Chrom'].to_csv("dng_chr_2.txt", index=False) #file for a list of␣
 ↪chrom numbers
# dngtrinuc['Pos'].to_csv("dng_pos_2.txt", index=False) #files that contains␣
 ↪a list of position numbers (lines on both files match up ie line 1 in chom␣
 ↪file links to line 1 pos file)
```

```
/dev/shm/jobs/33802852/ipykernel_53200/1762723996.py:19: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dngtrinuc.Alt_y[index] = s[0]
DNG finished
```

[5]:
```
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
######### uncomment for first running then Comment out for subequent runs to␣
 ↪save time
############################
# # upload the file that contains trinuc with reference allele
ref_trinuc = pd.read_csv("/nesi/project/nesi00187/erdue0/FinalFiles/
↪dng_triNucRef.txt", sep="\t", header=None ).astype(str)
print (ref_trinuc.head(5))

ref_chrPos= ref_trinuc[ref_trinuc[0].str.match('>')]     #getting only lines with␣
 ↪chr and pos on it
ref_chrPos  =ref_chrPos.rename(columns={0:  'chr:pos'})
ref_rtn = ref_trinuc[~ref_trinuc[0].str.startswith(('>'))]          #getting just lines␣
 ↪with the actual triNuc ("~" symbol for not)
ref_rtn= ref_rtn.rename(columns={0:"ref_tn"})

print(ref_chrPos.head(5))
print(ref_rtn.head(5))
# print(dngtrinuc.head(5))
# print (dngtrinuc)

_chrPosTrinuc =dngtrinuc.reset_index(drop=True).merge(ref_rtn.
 ↪reset_index(drop=True), left_index=True, right_index=True)  #magic code to␣
 ↪merge df's side by side
print(_chrPosTrinuc.head(5))

# ##########################################################################
```

```
print("starting for loop")
# For loop for getting the full triNuc ref>alt format and adding to df for
 ↪futher processing
# Get the full triNuc by substiting the ref allele for the alt allele in the
 ↪ref_tn column
for index, row in _chrPosTrinuc.iterrows():
    ref = _chrPosTrinuc.ref_tn.iloc[index]
    alt = [*_chrPosTrinuc.ref_tn.iloc[index]]
    alt[1] = _chrPosTrinuc.Alt_y.iloc[index]
    alt = ''.join(alt)
    # indexCombinedDF.at[index, 'software'] = val
    _chrPosTrinuc.at[index, 'triNuc'] = ref+ '>'+ alt
print(_chrPosTrinuc.head(5))
print ('formatting for loop finished')


# Note new Dataframe to use is _chrPosTrinuc as this has updated data
```

```
                          0
0    >chr1:9231709-9231711
1                      AGA
2  >chr1:41386458-41386460
3                      AAT
4  >chr1:77763773-77763775
                    chr:pos
0    >chr1:9231709-9231711
2  >chr1:41386458-41386460
4  >chr1:77763773-77763775
6  >chr1:79042521-79042523
8  >chr1:91423894-91423896
  ref_tn
1    AGA
3    AAT
5    CCC
7    CCA
9    GAC
  Chrom       Pos SAMPLE Ref  Alt_y  Genotype_y      DNMsc  Denovo-Posterior  \
0  chr1   9231710 trio6    G      T         0/1 -11.255200          0.000318
1  chr1  41386459 trio5    A      G         0/1  -0.003973          0.557895
2  chr1  77763774 trio5    C      G         0/1  -0.000016          0.387422
3  chr1  79042522 trio3    C      A         0/1  -1.989180          0.963735
4  chr1  91423895 trio1    A      T         0/1   0.000000          0.999994

  software  ref_tn
0     both     AGA
1     both     AAT
```

```
2       both      CCC
3       both      CCA
4       both      GAC
starting  for  loop
   Chrom       Pos SAMPLE Ref Alt_y Genotype_y       DNMsc Denovo-Posterior   \
0   chr1   9231710  trio6   G     T          0/1 -11.255200           0.000318
1   chr1  41386459  trio5   A     G          0/1  -0.003973           0.557895
2   chr1  77763774  trio5   C     G          0/1  -0.000016           0.387422
3   chr1  79042522  trio3   C     A          0/1  -1.989180           0.963735
4   chr1  91423895  trio1   A     T          0/1   0.000000           0.999994

   software ref_tn    triNuc
0      both    AGA  AGA>ATA
1      both    AAT  AAT>AGT
2      both    CCC  CCC>CGC
3      both    CCA  CCA>CAA
4      both    GAC  GAC>GTC
formatting for loop finished
```

[7]:
```
# # Code for writing out to a Bed file
_chrPosTrinuc["DNMsc"]    =    pd.to_numeric(dngtrinuc["DNMsc"])
_chrPosTrinuc.Pos = pd.to_numeric(_chrPosTrinuc.Pos)    #Using this dataframe has␣
 ↪it has adjusted/removed all INDEL data from DNG

unsortByDNG = _chrPosTrinuc.loc[_chrPosTrinuc.DNMsc == 0]   #0 is perfect score.␣
 ↪There are more than 100 of them
dng_lower = _chrPosTrinuc.loc[(_chrPosTrinuc.DNMsc <= -1) & (_chrPosTrinuc.
 ↪DNMsc >=-5)]
# print(dng_lower.triNuc)
# getting 100 candidates, an even spread from each proband sample of 0.0␣
 ↪(perfect) scoring DNG dnm candidates.
# Proband 38594368 signifcantly more 0.0 scoring candidates,
p = pd.concat([unsortByDNG.loc[unsortByDNG.SAMPLE == 'trio5'].sample(20),
               unsortByDNG.loc[unsortByDNG.SAMPLE == 'trio6'].sample(16),
               unsortByDNG.loc[unsortByDNG.SAMPLE == 'trio3'].sample(16),
               unsortByDNG.loc[unsortByDNG.SAMPLE == 'trio4'].sample(16),
               unsortByDNG.loc[unsortByDNG.SAMPLE == 'trio1'].sample(16),
               unsortByDNG.loc[unsortByDNG.SAMPLE    ==    'trio2'].sample(16)])

#code for getting randsom sample of lower scoring DNM within a set range (-1 ->␣
 ↪-5)
e =  pd.concat([dng_lower.loc[dng_lower.SAMPLE == 'trio5'].sample(20),
               dng_lower.loc[dng_lower.SAMPLE == 'trio6'].sample(16),
               dng_lower.loc[dng_lower.SAMPLE == 'trio3'].sample(16),
               dng_lower.loc[dng_lower.SAMPLE == 'trio4'].sample(16),
               dng_lower.loc[dng_lower.SAMPLE == 'trio1'].sample(16),
               dng_lower.loc[dng_lower.SAMPLE    ==    'trio2'].sample(16)])
```

```
# print(p)
#####################################################################
# # Commented out so as not to accidently change the randomly selected list
# bedout = open("dnmDng.bed","w")                ##change file name and input for␣
 ↪the different files
# for index, row in p.iterrows():
#       bedout.write("{}\t{}\t{}\n".format(row['Chrom'],  row['Pos'] -100,␣
 ↪row['Pos'] +100))

# bedout.close()
# # # # # print(sortByPed)
# # # # ###################################################################
# # # ##creating lists to easily find top candidates from each software for␣
 ↪validation
# bedout = open("dnmListDng.list","w")                ##change file name and input␣
 ↪for the different files
# for index, row in p.iterrows():
#       bedout.write("{}\t{}\t{}\t{}\t{}\t{}\t{}\n".format(row['Chrom'],␣
 ↪row['Pos'], row['SAMPLE'], row['Ref'], row['Alt_y'], row['DNMsc'], ␣
 ↪row['triNuc']))
# bedout.close()

#####################################################################
# # # Commented out so not recreating the same file again or changing random␣
 ↪selected  sample
# bedout = open("dnmDng_lower-1.bed","w")                ##change file name and␣
 ↪input for the different files
# for index, row in e.iterrows():
#       bedout.write("{}\t{}\t{}\n".format(row['Chrom'],  row['Pos'] -100,␣
 ↪row['Pos'] +100))

# bedout.close()
# # # # # print(sortByPed)
# # # # ###################################################################
# # # ##creating lists to easily find top candidates from each software for␣
 ↪validation
# bedout = open("dnmListDng_lower-1.list","w")                ##change file name␣
 ↪and input for the different files
# for index, row in e.iterrows():
#       bedout.write("{}\t{}\t{}\t{}\t{}\t{}\t{}\n".format(row['Chrom'],␣
 ↪row['Pos'], row['SAMPLE'], row['Ref'], row['Alt_y'], row['DNMsc'],␣
 ↪row['triNuc'] ))
# bedout.close()

# print(e.SAMPLE.value_counts())
```

```
# print (dng_lower)
```

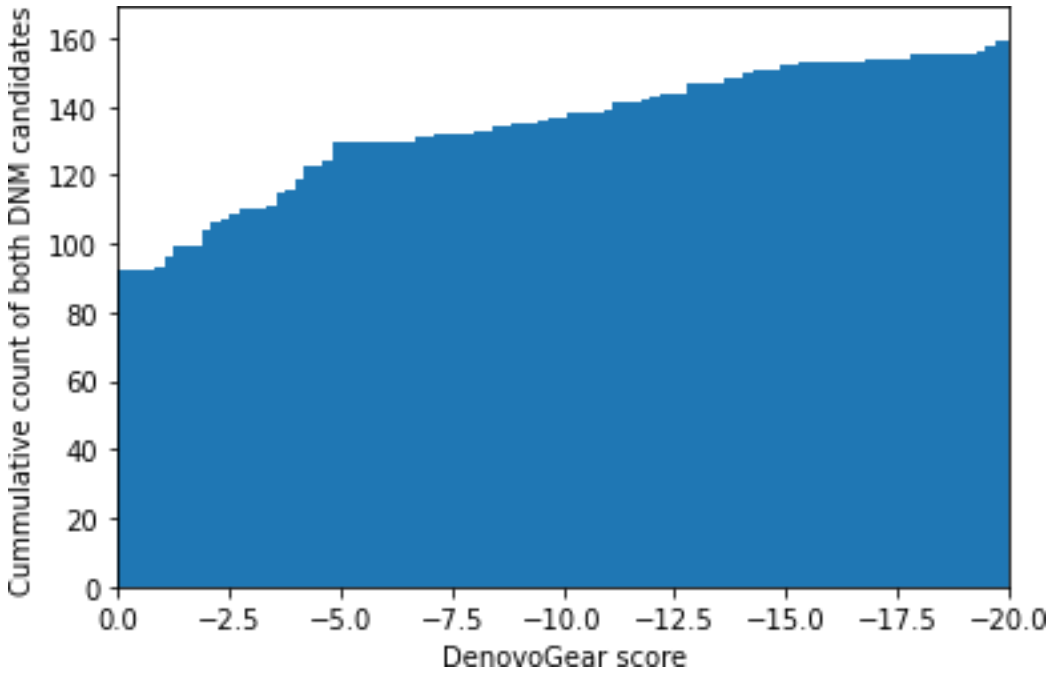
[8]:
```
# # data extraction
# # get count of each type of tri-nuc grouped by software tag
# # output data to be saved

# trioSample_ = indexCombinedDF.groupby(['SAMPLE'])['software'].value_counts() ␣
↪#group data by sample and software
# chrom_ =indexCombinedDF.groupby(['Chrom'])['software'].value_counts()        #group␣
↪data by Chrom and software (not super useful information just interesting)
dngTriCount = _chrPosTrinuc.triNuc.value_counts()
#  print('##################################################')
# # Prints whole output (without skipping any lines)
# print(dngTriCount.to_string()) #this line prints all out so it can be copied␣
↪to excel workbook for processing

#  print(dngTriCount.to_string().replace('        ', '\t'))    #replace sets up to␣
↪copy into excel easier by replaceing white space with \t
print(dngTriCount.head(5))

s = dngtrinuc.loc[dngtrinuc.DNMsc >= -5.5]
print(len(s))
print(len(dngtrinuc))
```

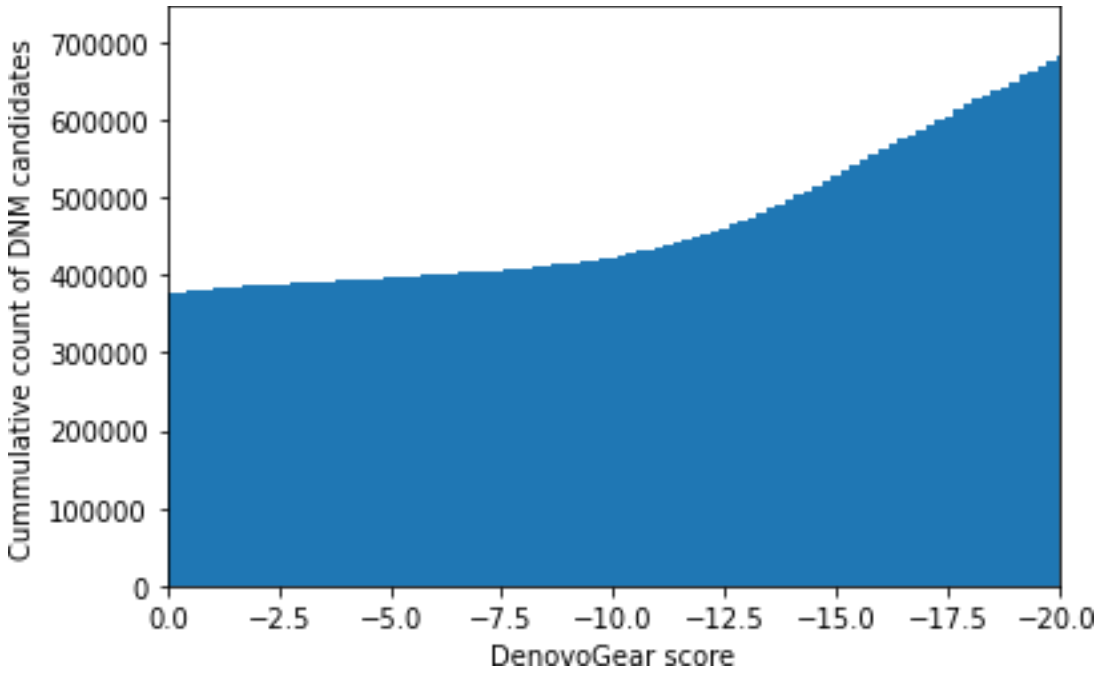
```
TGT>TAT    12904
ACA>ATA    12865
CGT>CAT    11865
ACG>ATG    11598
CAT>CGT    11473
Name: triNuc, dtype: int64
397856
711259
```

[9]:
```
# Threshold graphing
# The following code is to count the number of occurrences of scores within set␣
↪ranges
# Plot for DNMsc of DMN found by both software
fig, ax = plt.subplots()
bothData = dngtrinuc.loc[dngtrinuc.software == "both"]      #need the old data␣
↪frame as contains all the candidates for BOTH
plt.hist(bothData["DNMsc"], cumulative = -1, bins=100)  #still negative␣
↪cummulative plot for Dng data as 0 is the highest confidence interval and␣
↪largest number
ax.set_xlim(0, -20)
# plt.title('Cummulative count for DenovoGear scores for DNM Identified by Both␣
↪Software')
```

```
plt.xlabel("DenovoGear score")
plt.ylabel("Cummulative count of both DNM candidates")

#plot for DNMsc for DNM just found by dng
fig, ax = plt.subplots()
plt.hist(dngtrinuc["DNMsc"], cumulative = -1, bins=100) #inclusive of the both␣
↪data
ax.set_xlim(0, -20)
# plt.title('Cummulative count for DenovoGear scores for DNM Identified by␣
↪DenovoGear')
plt.xlabel("DenovoGear score")
plt.ylabel("Cummulative count of DNM candidates")
```

[9]: Text(0, 0.5, 'Cummulative count of DNM candidates')





[8]:
```
# ############################################################
# ## Code only needs to be run once and has been run. Can be used to update but␣
 ↪updates should be already included in main code
# # #code is written to add trinuc value to files of randomly selected␣
 ↪candidates after the fact
# igv_listUpper = pd.read_csv("/nesi/project/nesi00187/erdue0/dnmListDng.list",␣
 ↪sep="\t" ).astype(str)
# igv_listUpper.Pos = pd.to_numeric(igv_listUpper.Pos)   #converting to match␣
 ↪data in _chrPosTrinuc
# print (igv_listUpper.head(5))

# igv_listLower = pd.read_csv("/nesi/project/nesi00187/erdue0/
 ↪dnmListDng_lower-1.list", sep="\t", header=None ).astype(str)
# igv_listLower.columns = ['Chrom', 'Pos', 'Sample', 'Ref Allele', 'Alt␣
 ↪Allele',  'DNMsc']
# igv_listLower.Pos = pd.to_numeric(igv_listLower.Pos)   #converting to match␣
 ↪data in _chrPosTrinuc
# print (igv_listLower.head(5))


# #for each line in lower list search main data frame and add corresponding␣
 ↪trinuc to df
# lwListOut = open("dnmListDng_lower-1_trinuc.list","w")
# for index, row in igv_listLower.iterrows():
```

```
#       tn = _chrPosTrinuc.triNuc.loc[(_chrPosTrinuc.Chrom == igv_listLower.Chrom.
 ↪iloc[index]) & (_chrPosTrinuc.Pos == igv_listLower.Pos.iloc[index])]
#       # print(tn)
#       lwListOut.write("{}\t{}\t{}\t{}\t{}\t{}\t{}\n".format(row['Chrom'],␣
 ↪row['Pos'], row['Sample'], row['Ref Allele'], row['Alt Allele'],␣
 ↪row['DNMsc'], tn.iloc[0]))

# lwListOut.close()
# print('lower list updated')

# #for each line in upper list search main data frame and add corresponding␣
 ↪trinuc to df
# upListOut = open("dnmListDng_upper_trinuc.list","w")
# for index, row in igv_listUpper.iterrows():
#       tn = _chrPosTrinuc.triNuc.loc[(_chrPosTrinuc.Chrom == igv_listUpper.Chrom.
 ↪iloc[index]) & (_chrPosTrinuc.Pos == igv_listUpper.Pos.iloc[index])]
#       # print(tn)
#       upListOut.write("{}\t{}\t{}\t{}\t{}\t{}\t{}\n".format(row['Chrom'],␣
 ↪row['Pos'], row['Sample'], row['Ref Allele'], row['Alt Allele'],␣
 ↪row['DNMsc'], tn.iloc[0] ))

# upListOut.close()
# print('upper list updated')
```

[11]:
```
###
#testing block of code
e =dngtrinuc.loc[(dngtrinuc.SAMPLE == 'trio1') ]
f =dngtrinuc.loc[(dngtrinuc.SAMPLE == 'trio2') ]
c =dngtrinuc.loc[ (dngtrinuc.SAMPLE == 'trio3') ]
d =dngtrinuc.loc[(dngtrinuc.SAMPLE == 'trio4') ]
a =dngtrinuc.loc[ (dngtrinuc.SAMPLE == 'trio5') ]
b =dngtrinuc.loc[ (dngtrinuc.SAMPLE == 'trio6') ]

# s =dngtrinuc.loc[(dngtrinuc.DNMsc <= -1) & (dngtrinuc.DNMsc >= -5) &␣
 ↪(dngtrinuc.SAMPLE == 'trio1') ]
# a =dngtrinuc.loc[(dngtrinuc.DNMsc <= -1) & (dngtrinuc.DNMsc >= -5) &␣
 ↪(dngtrinuc.SAMPLE == 'trio2') ]
 # b =dngtrinuc.loc[(dngtrinuc.DNMsc <= -1) & (dngtrinuc.DNMsc >= -5) &␣
 ↪(dngtrinuc.SAMPLE == 'trio3') ]
# c =dngtrinuc.loc[(dngtrinuc.DNMsc <= -1) & (dngtrinuc.DNMsc >= -5) &␣
 ↪(dngtrinuc.SAMPLE == 'trio4') ]
 # d =dngtrinuc.loc[(dngtrinuc.DNMsc <= -1) & (dngtrinuc.DNMsc >= -5) &␣
 ↪(dngtrinuc.SAMPLE == 'trio5') ]
# e =dngtrinuc.loc[(dngtrinuc.DNMsc <= -1) & (dngtrinuc.DNMsc >= -5) &␣
 ↪(dngtrinuc.SAMPLE == 'trio6') ]
```

```
e =dngtrinuc.loc[(dngtrinuc.SAMPLE == 'trio1') & (dngtrinuc.DNMsc == 0)]
f =dngtrinuc.loc[(dngtrinuc.SAMPLE == 'trio2') & (dngtrinuc.DNMsc == 0)]
c =dngtrinuc.loc[(dngtrinuc.SAMPLE == 'trio3') & (dngtrinuc.DNMsc == 0)]
d =dngtrinuc.loc[(dngtrinuc.SAMPLE == 'trio4') & (dngtrinuc.DNMsc == 0)]
a =dngtrinuc.loc[(dngtrinuc.SAMPLE == 'trio5') & (dngtrinuc.DNMsc == 0)]
b =dngtrinuc.loc[(dngtrinuc.SAMPLE == 'trio6') & (dngtrinuc.DNMsc == 0)]

print(len(a))
print(len(b))
print(len(c))
print(len(d))
print(len(e))
print(len(f))
print(len(f) + len(a) + len(b) + len(c) + len(d) + len(e))
# print(len(dngtrinuc))

print(len(dngtrinuc))
```

```
88584
435
55
30
56
41
89201
711259
```

[ ]:

Appendix C: Jupyter NoteBook for PedFilter and Both Software Packages

[1]:
```
#testing import statements for whether nesi has installed them or not
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import csv
print ("Library imports successful")

#loading DNG data file into dataframe for processing and manipulation
denovogearDF = pd.read_csv("/nesi/project/nesi00187/erdue0/FinalFiles/
↪denovogear_noChrX_v2.csv", sep=",", header=0).astype(str)
# print(len(denovogearDF))
# print(denovogearDF.head(5)) #print to see has loaded correctly

#loading pedFilter data file into separate dataframe for processing and␣
↪manipulation
print("#################################################################################")
pedfilterDF = pd.read_csv("/nesi/project/nesi00187/erdue0/FinalFiles/
↪trioPedFilterCombine_v2.csv", sep=",", header=0).astype(str)
pedfilterDF.Genotype = pedfilterDF['Genotype'].str.replace('\'','')        #cleaning␣
↪input data of random unnecessary symbols
#removing unneccessary lines from where separate chrom files were appended into␣
↪1 file
pedfilterDF = pedfilterDF.loc[(pedfilterDF['Chrom'] != 'nan') &␣
↪(pedfilterDF['Chrom'] != 'Chrom') & (pedfilterDF['Chrom'] != 'done') &␣
↪(pedfilterDF['Pos'] != 'nan')]

print(len(pedfilterDF))
# print(pedfilterDF.head(5))    #print to see has loaded correctly

##############
##Error checking unwanted data was filtered out correctly from pedfilter and␣
↪denovogear dataframe
# Testing both dataframes for any genotypes that are not 0/1 (should be none)
print('################################')
print(denovogearDF.loc[denovogearDF['Genotype'] != '0/1'])
print(pedfilterDF.loc[pedfilterDF['Chrom']  ==  'Chrom'])
```

```
##############################
#Combining DNG and PEDfilter dataframes into sinlge entity. Outer join merges␣
 ↪data that was found in both packages and keeps the data that was only found␣
 ↪in 1
indexCombinedDF = pd.merge(pedfilterDF, denovogearDF,␣
↪on=["Chrom","Pos","SAMPLE",    "Ref"],    how="outer")
print(len(indexCombinedDF))
print(indexCombinedDF.head(5))
print(len(indexCombinedDF)) # to see that some data points have merged (dnm␣
 ↪candidates found by both software)
##################################
#Error checking combined dataframe before further processing
#checking there are no chrom data points that hold "Chrom" instead of chrom␣
 ↪number
print (indexCombinedDF.loc[indexCombinedDF["Chrom"] == "Chrom"])
# # used to test of any null variable points where there shouldn't be
#  indexCombinedDF = indexCombinedDF.loc[indexCombinedDF['Chrom'] != 'nan']
```

```
Library imports successful
################################################################################
##############################
3134
##############################
Empty DataFrame
Columns: [Unnamed: 0, Chrom, Pos, Ref, Alt, SAMPLE, Genotype, DNMsc]
Index: []
Empty DataFrame
Columns: [Unnamed: 0, Chrom, Pos, rsID, TRI-NUC, Ref, Alt, QUAL, SAMPLE,
Genotype, PLs, Denovo-Posterior, Probs(mend,denovo,bad), MichelsCriteria,
Source, Phase, Denovo-S|D, Haps-S|D, Ances, Pars, Children, Desc, ExFam, Pop,
PopFreq, Support_Ratio, Offspring-Support-Ratio, Class, Proband_dif_kids,
Proband_dif_pop, Sire_Dif_Kids, Dam_dif_kids, _Sire_dif_pop, Dam_dif_Pop, PopAD,
Proband, Sire, gSire, gDam, Dam, gSire.1, gDam.1, PopAB, PhaseInfo, Unnamed: 43]
Index: []

[0 rows x 45 columns]
950455
   Unnamed: 0_x Chrom      Pos rsID     TRI-NUC    Ref Alt_x    QUAL SAMPLE \
0             0  chr1   926385    .     CCA>CAA      C     A   11.62   trio3
1             1  chr1  3035478    .     ACA>AAA      C     A   112.8   trio2
2             2  chr1  3097802    .     TGG>TTG      G     T    24.6   trio3
3             3  chr1  3374744    .     AGA>AAA      G     A  375.13   trio2
4             4  chr1  3955781    . AAG>INDEL  AGAGG     A  316.07   trio1

  Genotype_x    ...                 Dam gSire.1 gDam.1 PopAB         PhaseInfo   \
0        0/1    ...    (21,0)  0,57,855     nan    nan  9232               0 0.0
```

```
1          0/1  ...  (26,0)  0,66,1013      nan     nan  8572  10 0.0011652296
2          0/1  ...   (14,0)  0,39,585       nan     nan  8846              0 0.0
3          0/1  ...   (32,0)  0,14,954       nan     nan  9033   5  5.5321975E-4
4          0/1  ...  (37,0)  0,72,1080      nan     nan  8826              0 0.0

  Unnamed: 43 Unnamed: 0_y Alt_y Genotype_y DNMsc
0         nan          NaN   NaN        NaN   NaN
1         nan          NaN   NaN        NaN   NaN
2         nan          NaN   NaN        NaN   NaN
3         nan          NaN   NaN        NaN   NaN
4         nan          NaN   NaN        NaN   NaN

[5 rows x 49 columns]
950455
Empty DataFrame
Columns: [Unnamed: 0_x, Chrom, Pos, rsID, TRI-NUC, Ref, Alt_x, QUAL, SAMPLE,
Genotype_x, PLs, Denovo-Posterior, Probs(mend,denovo,bad), MichelsCriteria,
Source, Phase, Denovo-S|D, Haps-S|D, Ances, Pars, Children, Desc, ExFam, Pop,
PopFreq, Support_Ratio, Offspring-Support-Ratio, Class, Proband_dif_kids,
Proband_dif_pop, Sire_Dif_Kids, Dam_dif_kids, _Sire_dif_pop, Dam_dif_Pop, PopAD,
Proband, Sire, gSire, gDam, Dam, gSire.1, gDam.1, PopAB, PhaseInfo, Unnamed: 43,
Unnamed: 0_y, Alt_y, Genotype_y, DNMsc]
Index: []

[0 rows x 49 columns]
```

[2]:
```
# # Code to add Column tagging if the DNM candidate was found by PedFilter, DNM_
 ↪or both
# Line to apply method to Df
indexCombinedDF['software'] = np.nan # fullCombinedDF['software'] = np.nan #_
 ↪cpCombinedDF['software'] = np.nan

# # header['software'] = np.nan
# indexCombinedDF = indexCombinedDF.reset_index() # fullCombinedDF =_
 ↪fullCombinedDF.reset_index() # cpCombinedDF = cpCombinedDF.reset_index()

# header = header.reset_index()
print (indexCombinedDF.head())

#for loop for indexCombined (samp+ref) Df to add
for index, row in indexCombinedDF.iterrows():
    val = ''
    if (pd.notnull(row[11]) & pd.notnull(row['DNMsc']) ):
        val = 'both'
    elif (pd.notnull(row[11])):
        val = 'ped'
    elif (pd.notnull(row['DNMsc'])):
```

```
            val = "dng"
        else:
            print("else entered")
            val = "error"
        indexCombinedDF.at[index, 'software'] = val
print('first  for  loop  done  (1/3)')

print('verified  identifier')
print (indexCombinedDF.head())
print('and  tail  for  testing')
print (indexCombinedDF.tail())
```

```
   Unnamed: 0_x Chrom      Pos rsID     TRI-NUC    Ref Alt_x     QUAL SAMPLE \
0             0  chr1   926385    .    CCA>CAA      C     A    11.62  trio3
1             1  chr1  3035478    .    ACA>AAA      C     A    112.8  trio2
2             2  chr1  3097802    .    TGG>TTG      G     T     24.6  trio3
3             3  chr1  3374744    .    AGA>AAA      G     A   375.13  trio2
4             4  chr1  3955781    .  AAG>INDEL  AGAGG     A   316.07   trio1

   Genotype_x  ... gSire.1 gDam.1 PopAB         PhaseInfo Unnamed: 43  \
0         0/1  ...     nan    nan  9232               0 0.0          nan
1         0/1  ...     nan    nan  8572  10 0.0011652296          nan
2         0/1  ...     nan    nan  8846               0 0.0          nan
3         0/1  ...     nan    nan  9033    5 5.5321975E-4          nan
4         0/1  ...     nan    nan  8826               0 0.0          nan

   Unnamed: 0_y Alt_y Genotype_y DNMsc software
0           NaN   NaN        NaN   NaN      NaN
1           NaN   NaN        NaN   NaN      NaN
2           NaN   NaN        NaN   NaN      NaN
3           NaN   NaN        NaN   NaN      NaN
4           NaN   NaN        NaN   NaN      NaN

[5 rows x 50 columns]
first for loop done (1/3)
verified  identifier
   Unnamed: 0_x Chrom      Pos rsID     TRI-NUC    Ref Alt_x     QUAL SAMPLE \
0             0  chr1   926385    .    CCA>CAA      C     A    11.62  trio3
1             1  chr1  3035478    .    ACA>AAA      C     A    112.8  trio2
2             2  chr1  3097802    .    TGG>TTG      G     T     24.6  trio3
3             3  chr1  3374744    .    AGA>AAA      G     A   375.13   trio2
4             4  chr1  3955781    .  AAG>INDEL  AGAGG     A   316.07   trio1

   Genotype_x  ... gSire.1 gDam.1 PopAB         PhaseInfo Unnamed: 43  \
0         0/1  ...     nan    nan  9232               0 0.0          nan
1         0/1  ...     nan    nan  8572  10 0.0011652296          nan
2         0/1  ...     nan    nan  8846               0 0.0          nan
```

```
3         0/1   ...      nan     nan  9033    5  5.5321975E-4          nan
4         0/1   ...      nan     nan  8826              0 0.0          nan

  Unnamed: 0_y Alt_y Genotype_y DNMsc software
0          NaN   NaN        NaN   NaN      ped
1          NaN   NaN        NaN   NaN      ped
2          NaN   NaN        NaN   NaN      ped
3          NaN   NaN        NaN   NaN      ped
4          NaN   NaN        NaN   NaN      ped

[5 rows x 50 columns]
and tail for testing
       Unnamed: 0_x  Chrom       Pos rsID TRI-NUC  Ref Alt_x QUAL SAMPLE  \
950450           NaN  chr29  51062394  NaN     NaN  CCC   NaN  NaN  trio5
950451           NaN  chr29  51070218  NaN     NaN    A   NaN  NaN  trio3
950452           NaN  chr29  51079711  NaN     NaN    T   NaN  NaN  trio3
950453           NaN  chr29  51081000  NaN     NaN    G   NaN  NaN  trio4
950454           NaN  chr29  51096068  NaN     NaN    A   NaN  NaN  trio2

       Genotype_x  ... gSire.1 gDam.1 PopAB PhaseInfo Unnamed: 43  \
950450        NaN  ...     NaN    NaN   NaN       NaN         NaN
950451        NaN  ...     NaN    NaN   NaN       NaN         NaN
950452        NaN  ...     NaN    NaN   NaN       NaN         NaN
950453        NaN  ...     NaN    NaN   NaN       NaN         NaN
950454        NaN  ...     NaN    NaN   NaN       NaN         NaN

       Unnamed: 0_y          Alt_y Genotype_y    DNMsc software
950450      5447033  CCTCC,CCCTCC        0/1 -18.8813      dng
950451      5447045              T        0/1 -20.2604      dng
950452      5447092              C        0/1 -14.5504      dng
950453      5447113              T        0/1  -17.037      dng
950454      5447148              G        0/1 -10.8551      dng

[5 rows x 50 columns]
```

[3]:
```
# # Chunk of code responsible for filtering out PedFilter INDEL data
##filtering PedFilter data for INDEL data
pedtrinuc = indexCombinedDF.loc[(indexCombinedDF["software"] == "ped") |
↪(indexCombinedDF["software"] == "both")]
pedtrinuc = pedtrinuc[["Chrom","Pos", "SAMPLE", "Ref", "Alt_x", "Genotype_x",
↪"Denovo-Posterior", "TRI-NUC", "software"]]
pedtrinuc["Denovo-Posterior"] = pd.to_numeric(pedtrinuc["Denovo-Posterior"])
pedtrinuc = pedtrinuc.loc[(pedtrinuc.Ref == "A") | (pedtrinuc.Ref == "T") |
↪(pedtrinuc.Ref == "G") | (pedtrinuc.Ref == "C")]  #good filtering for
↪triNuc data but keep elsewhere for other analysis

#for loop for getting base represented by the 0/1 genotypes in this dataframe
```

```
  for index, row in pedtrinuc.iterrows():
   if (',' in pedtrinuc.Alt_x[index]):
          # print("TRUE")
          s    =    pedtrinuc.Alt_x[index].split(',')
          # print(s)
          pedtrinuc.Alt_x[index] = s[0]

pedtrinuc = pedtrinuc.loc[(pedtrinuc.Alt_x == 'A') | (pedtrinuc.Alt_x == 'T') |␣
 ↪(pedtrinuc.Alt_x == 'G') | (pedtrinuc.Alt_x == 'C')]  #keep after sorting␣
 ↪for SNPs with mulitple genotypes
pedtrinuc = pedtrinuc.loc[~pedtrinuc['TRI-NUC'].str.contains('INDEL')]          #make␣
 ↪sure no indels are left
print ('finished')
```

```
/dev/shm/jobs/34156464/ipykernel_125820/2225459164.py:14:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  pedtrinuc.Alt_x[index] = s[0]
finished
```

[4]:
```
# code of writing out to a BED file
pedtrinuc['Denovo-Posterior']    =    pd.to_numeric(pedtrinuc['Denovo-Posterior'])
pedtrinuc.Pos    =    pd.to_numeric(pedtrinuc['Pos'])
pedSort = pedtrinuc.sort_values(by='Denovo-Posterior', ascending=False)  #by␣
 ↪sorting the whole dataframe data points tagged ped and both will be included
sortByPed = pedSort.head(100)
#########################################################################
# # Code for writing out bed file for Samtools merge command to get data out␣
 ↪for IGV validation
# # bedout = open("dnmPed.bed","w")
# # for index, row in sortByPed.iterrows():
# #       bedout.write("{}\t{}\t{}\n".format(row['Chrom'], row['Pos'] -100,␣
 ↪row['Pos'] +100))

# #       # print(row['Chrom'])
# #  bedout.close()
# # # # # print(pedData.head())
# # # # # # print(sortByPed)
# # # # ########################################################################
# # ##creating lists to easily find top candidates from each software for␣
 ↪validation
# bedout = open("dnmListPed.list","w")                ##change file name and input␣
 ↪for the different files
```

```
# for index, row in sortByPed.iterrows():
#       bedout.writepedtrinuct{}\t{pedtrinuct{}\t{}\t{}\n".format(row['Chrom'],␣
 ↪row['Pos'], row['SAMPLE'], row['Ref'], row['Alt_x'],␣
 ↪row['Denovo-Posterior'], row['TRI-NUC'] ))

#       # print(row['Chrom'])
#  bedout.close()

# #␣
 ↪#############################################################################################
# # output 100 other ped candidates at a lower range
#  pedLower = pedSort.loc[pedtrinuc['Denovo-Posterior'] <= 0.90].head(100)
# print(pedLower.head(5))

# # ##creating lists to easily find top candidates from each software for␣
 ↪validation
# bedout = open("dnmListPed_lowerRange90.list","w")                ##change file␣
 ↪name and input for the different files
# for index, row in pedLower.iterrows():
#       bedout.write("{}\t{}\t{}\t{}\t{}\t{}\t{}\n".format(row['Chrom'],␣
 ↪row['Pos'], row['SAMPLE'], row['Ref'], row['Alt_x'],␣
 ↪row['Denovo-Posterior'],  row['TRI-NUC']))

#       # print(row['Chrom'])
#  bedout.close()
# ###########################################################################
# # Code for writing out bed file for Samtools merge command to get data out␣
 ↪for IGV validation
# bedout = open("dnmPed_lowerRange90.bed","w")              ##change file name␣
 ↪and input for the different files
# for index, row in pedLower.iterrows():
#       bedout.write("{}\t{}\t{}\n".format(row['Chrom'], row['Pos'] -100,␣
 ↪row['Pos'] +100))

#       # print(row['Chrom'])
#  bedout.close()
# # # # print(pedData.head())
# # # # # # print(sortByPed)
#␣
 ↪#############################################################################################
# # Code for gettting data for ALL of the candidates found by BOTH software
bothcandidates = indexCombinedDF.loc[indexCombinedDF["software"] == "both"]
print(bothcandidates.head(10))
# print(len(bothcandidates))
```

```
# # ##creating lists to easily find top candidates from each software for␣
 ↪validation
# bothout = open("dnmListBoth.list","w")                ##change file name and␣
 ↪input for the different files
# for index, row in bothcandidates.iterrows():
#       bothout.write("{}\t{}\t{}\t{}\t{}\t{}\t{}\n".format(row['Chrom'],␣
 ↪row['Pos'], row['SAMPLE'], row['Ref'], row['Alt_x'],␣
 ↪row['Denovo-Posterior'],  row['TRI-NUC']))

#       # print(row['Chrom'])
#  bothout.close()

#␣
 ↪#############################################################################
# # Code for getting bed file for Both candidates
# bedout = open("dnmBoth.bed","w")
# for index, row in bothcandidates.iterrows():
#       bedout.write("{}\t{}\t{}\n".format(row['Chrom'],  row['Pos'] -100,␣
 ↪row['Pos'] +100))

#       # print(row['Chrom'])
#  bedout.close()
# # # # print(pedData.head())
# # # # # # print(sortByPed)
```

	Unnamed: 0_x	Chrom	Pos	rsID	TRI-NUC	Ref	Alt_x	QUAL	SAMPLE	\
12	12	chr1	9231710	.	AGA>ATA	G	T	709.24	trio6	
44	44	chr1	41386459	.	AAT>AGT	A	G	125.61	trio5	
79	79	chr1	77763774	.	CCC>CGC	C	G	699.17	trio5	
83	83	chr1	79042522	.	CCA>CAA	C	A	60.65	trio3	
101	101	chr1	91423895	.	GAC>GTC	A	T	709.57	trio1	
120	120	chr1	117712596	.	AGG>ATG	G	T	25.57	trio3	
125	125	chr1	120357361	.	CTA>CCA	T	C	765.3	trio5	
132	132	chr1	124968402	.	AGA>AAA	G	A	414.64	trio5	
134	134	chr1	127958490	.	GTG>GAG	T	A	567.2	trio5	
193	196	chr2	15062455	.	AGG>AAG	G	A	806.15	trio4	

	Genotype_x	…	gSire.1	gDam.1	PopAB	PhaseInfo	Unnamed: 43	Unnamed: 0_y	\
12	0/1	…	nan	nan	8970	0	0.0	nan	28898	
44	0/1	…	nan	nan	8909	0	0.0	nan	108161	
79	0/1	…	nan	nan	9167	0	0.0	nan	181793	
83	0/1	…	nan	nan	8983	0	0.0	nan	182999	
101	0/1	…	nan	nan	8958	0	0.0	nan	198071	
120	0/1	…	nan	nan	9423	0	0.0	nan	245061	
125	0/1	…	nan	nan	8605	0	0.0	nan	249329	
132	0/1	…	nan	nan	9309	0	0.0	nan	258418	
134	0/1	…	nan	nan	8939	0	0.0	nan	262344	

```
193          0/1   ...      nan      nan  8790       0  0.0           nan          351196

     Alt_y Genotype_y          DNMsc  software
12       T        0/1       -11.2552      both
44       G        0/1     -0.00397317     both
79       G        0/1     -1.58488e-05    both
83       A        0/1       -1.98918      both
101      T        0/1            0.0      both
120      T        0/1       -4.16038      both
125      C        0/1     -1.99526e-06    both
132      A        0/1            0.0      both
134      A        0/1     -2.51177e-12    both
193      A        0/1    -0.000125885     both

[10 rows x 50 columns]
```

[6]:
```
# # output 100 other ped candidates at a lower range
# pedLower = pedSort.loc[pedtrinuc['Denovo-Posterior'] <= 0.90].head(100)
# print(pedLower.head(5))

# # ##creating lists to easily find top candidates from each software for␣
 ↪validation
# bedout = open("dnmListPed_lowerRange90.list","w")                ##change file␣
 ↪name and input for the different files
# for index, row in pedLower.iterrows():
#       bedout.write("{}\t{}\t{}\t{}\t{}\t{}\t{}\n".format(row['Chrom'],␣
 ↪row['Pos'], row['SAMPLE'], row['Ref'], row['Alt_x'],␣
 ↪row['Denovo-Posterior'],  row['TRI-NUC']))

#       # print(row['Chrom'])
# bedout.close()
```

[7]:
```
#Code for extracting trinuc data for ped (and both using the ped tag)
# trinuc_pedBoth = p.groupby(['software'])['TRI-NUC'].value_counts()
trinuc_ped = pedtrinuc["TRI-NUC"].value_counts()
bothData = pedtrinuc.loc[pedtrinuc.software == "both"]
trinuc_both = bothData["TRI-NUC"].value_counts()
print(trinuc_both.head(3))
print(trinuc_ped.head(3))
# # Prints whole output (without skipping any lines)
# print (trinuc_both.to_string())
# print(trinuc_ped.to_string())
# print(trinuc_ped.to_string().replace('   ', '\t'))
```

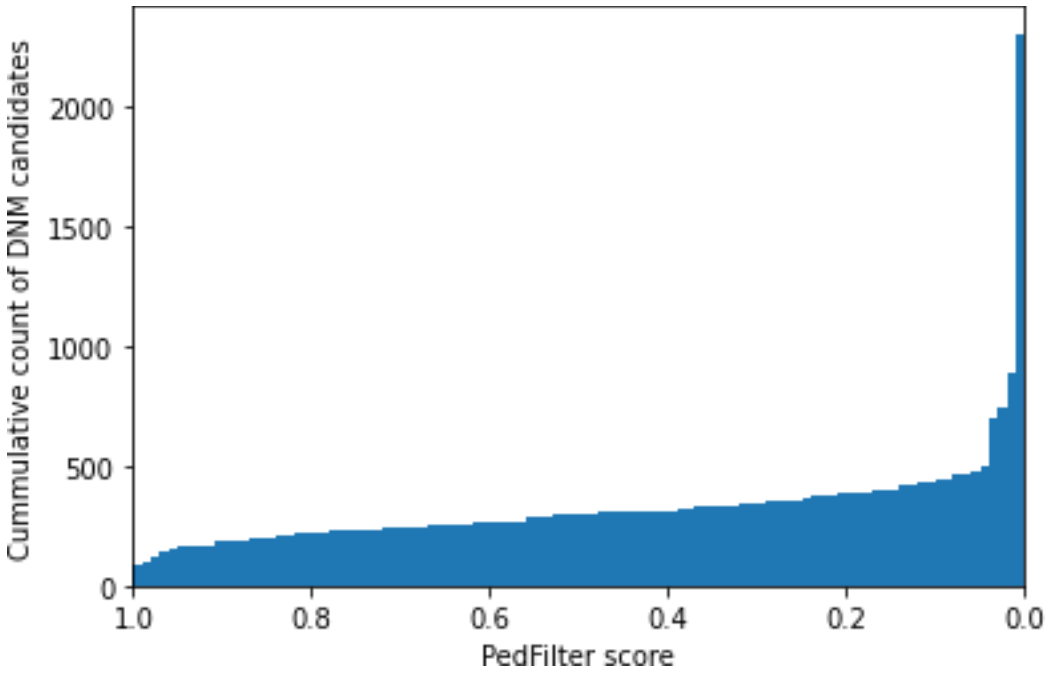
```
TGG>TTG    8
CCC>CAC    6
TGT>TTT    4
```

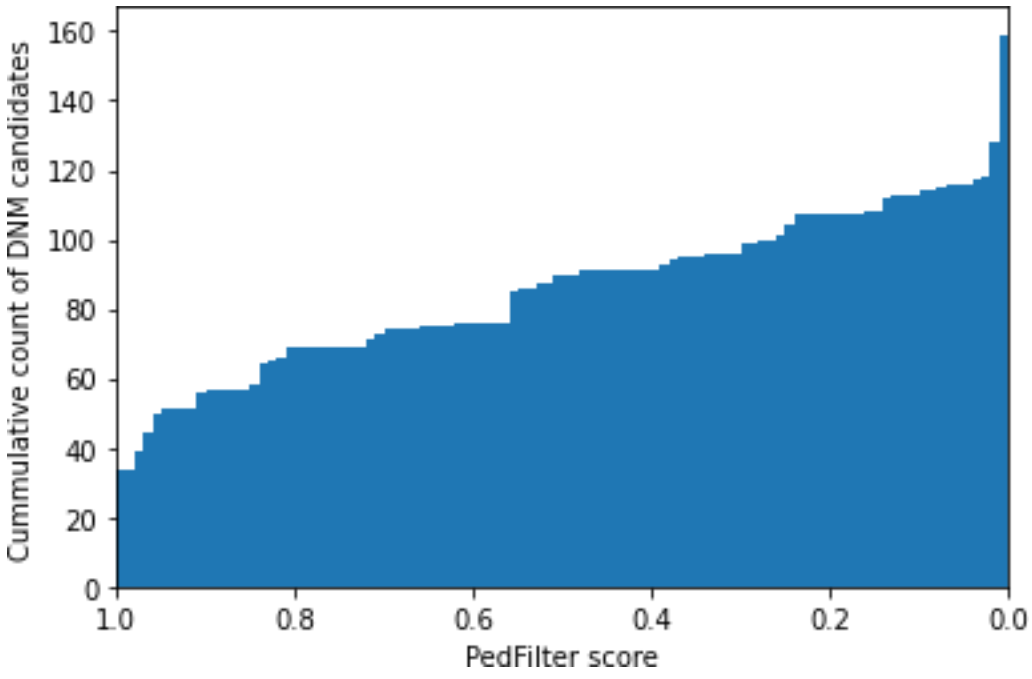
```
Name: TRI-NUC, dtype: int64
CAC>CCC  152
GTT>GGT    132
GTG>GGG    129
Name: TRI-NUC, dtype: int64
```

[8]:
```
#Plot for PedData threshold testing
bothData = pedtrinuc.loc[pedtrinuc.software == "both"]

fig, ax = plt.subplots()
print("starting  graphs")
ax.hist(pedtrinuc["Denovo-Posterior"], cumulative =-1, bins=100) #negative␣
↪cummulative graph starts with largest interval and cummulates from there

ax.set_xlim(1, 0)   #This is importants as for pedfilter 1 is the best and the␣
↪closer to 0 the less confident of a true DNM program is
# plt.title('Cummulative count for Denovo-Posterior scores in PedFilter')
plt.xlabel("PedFilter  score")
plt.ylabel("Cummulative count of DNM candidates")
# plot for pedFilter data from DNM found by both software
fig, ax = plt.subplots()
plt.hist(bothData["Denovo-Posterior"], cumulative = -1, bins=100)
ax.set_xlim(1, 0)
# plt.title('Cummulative count for PedFilter scores for DNM Identified by Both␣
↪Software')
plt.xlabel("PedFilter  score")
plt.ylabel("Cummulative count of DNM candidates")
```

```
starting  graphs
```

[8]:
```
Text(0, 0.5, 'Cummulative count of DNM candidates')
```





[7]:
```
a = pedtrinuc.loc[(pedtrinuc.SAMPLE == 'trio1')]
b = pedtrinuc.loc[(pedtrinuc.SAMPLE == 'trio2')]
```

```
c = pedtrinuc.loc[(pedtrinuc.SAMPLE == 'trio3')]
d = pedtrinuc.loc[(pedtrinuc.SAMPLE == 'trio4')]
e = pedtrinuc.loc[(pedtrinuc.SAMPLE == 'trio5')]
f = pedtrinuc.loc[(pedtrinuc.SAMPLE == 'trio6')]

print(len(a))
print(len(b))
print(len(c))
print(len(d))
print(len(e))
print(len(f))
```

```
172
1133
693
128
124
55
```

[ ]: