How to Evaluate Roche Target Enrichment Data for Somatic Variant Research

1. Overview

Analysis of Roche TE (AVENIO Edge or KAPA Target Enrichment) experimental data sequenced on an Illumina sequencing system is most frequently performed using a variety of publicly available, open-source analysis tools.

The usage examples described here have been used effectively in our hands. *Please note that publicly available, open-source software tools may change and that such change is not under the control of Roche. Therefore Roche does not warrant and cannot be held liable for the results obtained when using the third party tools described herein. Roche does not provide direct analysis support or service for these or any other third party tools. Please refer to the authors of each tool for support and documentation.*

The typical variant calling analysis workflow consists of sequencing read quality assessment, read filtering, mapping against the reference genome, duplicate removal, coverage statistic assessment, variant calling, and variant filtering. At most of these steps, a variety of tools can be utilized. This document shows how to use a selection of the available tools to perform Roche TE data analysis for somatic variant applications, but other analysis workflows can also be used.

This document will enable readers with bioinformatics experience to understand the basic analysis workflow in use at Roche to assess capture performance. The reader should carefully consider additional options when deciding the most appropriate workflow for their research.

2. Solution

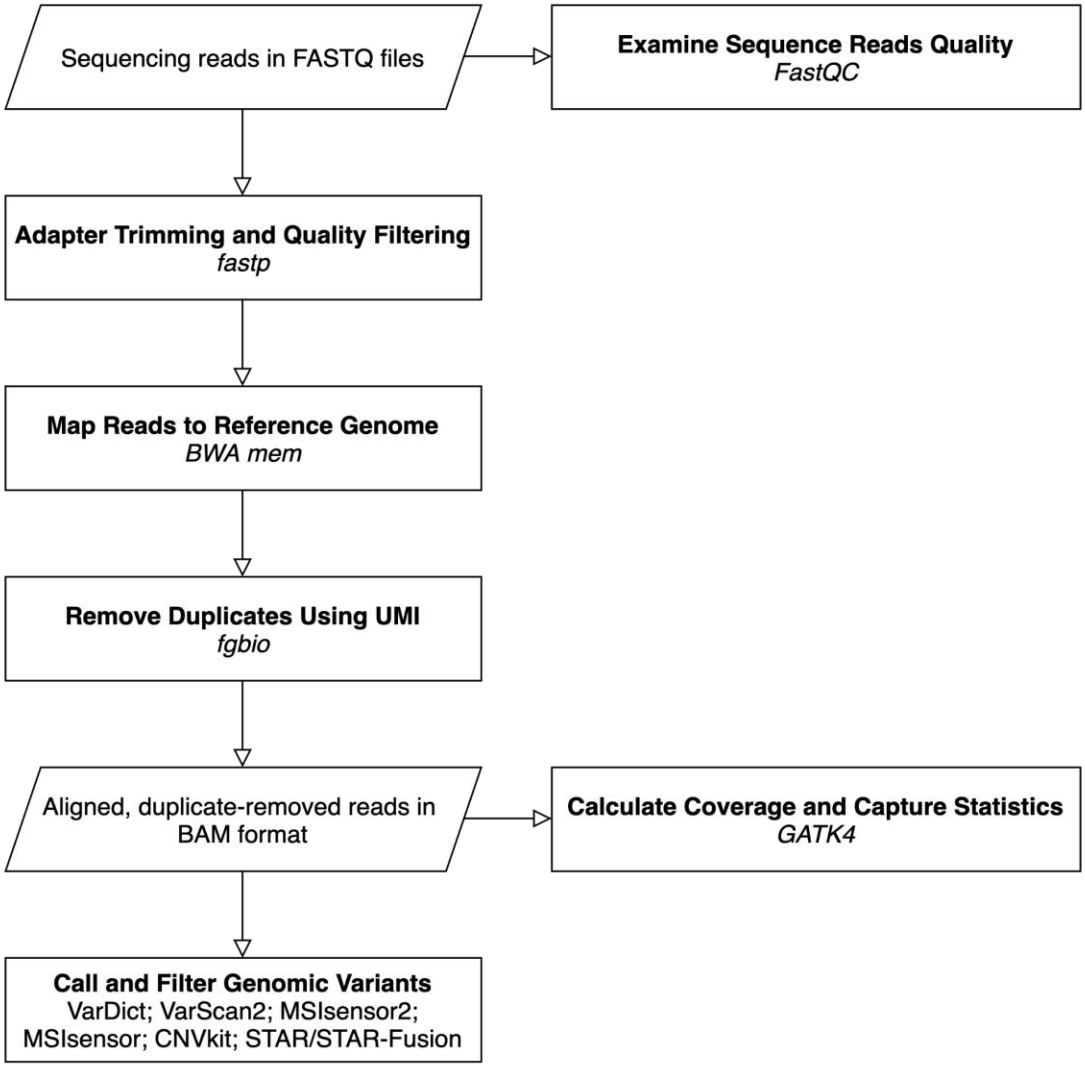

Figure 1. Schematic of basic analysis workflow.

Free and open source third-party tools are available for converting raw sequencing data into appropriate file formats, mapping reads to a reference sequence, evaluating sequencing quality, and analyzing variant calls. This white paper describes a number of steps and mini-workflows that use such third-party tools, which can be combined together into a variety of data analysis workflows.

Ideally, you should develop a workflow appropriate for your experimental data using benchmark/control samples that contain somatic variants at different levels.

Note that where the text "SAMPLE" appears throughout examples shown here, you should replace it with a unique sample name. Similarly, replace "DESIGN" with the name of the target enrichment design that matches the design files supplied by Roche. "NumProcessors" should be replaced with the number of CPU cores available.

Replace "/path/to/…" in the examples with a valid path on your system. The current directory is assumed to be the location of all input files, and will also be the location of output files and report files. Some of the tools described in

this document require execution of a .jar file by calling Java. One exception is GATK, which requires Java but is executed through a wrapper. If Java 1.8 is not the default version on your system, you will need to execute the GATK .jar file using a direct path to Java 1.8 instead.

Type the entire command shown for each step on a single line, despite the way it appears on the printed page. There should be no spaces within a file path, but there must be spaces before and after each option. Due to idiosyncrasies in most if not all PDF viewers, the underscores in command line examples in this document may not display properly at all zoom percentages. One way to confirm whether or not underscores are present is to print the page. Alternatively, try temporarily switching to a very high zoom percentage (*e.g.*, 400%).

Examples included in this document show how to perform the analysis with paired end Illumina sequencing reads. Many of the tools work with single end reads if paired end reads are not available, though input and output formats may vary. Note that using single end reads will artificially increase duplicate rate due to decreased ability to resolve a unique fragment from the library.

Tools Overview

Package (version)	Tool	Function as used in this document
BWA (0.7.17)	`index`	Generate an indexed genome from FASTA sequence.
	`mem`	Map sequencing reads to an indexed genome.
FastQC (0.11.9)	`Fastqc`	Assess sequencing read quality (per-base quality plot).
GATK4* (4.2.0.0)	`BedToIntervalList`	Convert BED file to Genomic Interval List format.
	`CollectHsMetrics`	Assess performance of a target enrichment experiment based on mapped reads.
	`CollectAlignmentSummaryMetrics`	Report mapping metrics for a BAM file.
	`CollectInsertSizeMetrics`	Estimate and plot insert size distribution.
	`CountReads`	Count the number of sequencing reads overlapping target regions.
	`CreateSequenceDictionary`	Generate a sequence dictionary (.dict) for the reference genome.
	`FastqToSam`	Converts a FASTQ file to an unaligned BAM or SAM file.
	`FixMateInformation`	Clean up paired read information.
	`IndexFeatureFile`	Generate an index (.idx) for a VCF file.
	`MarkDuplicates`	Count the number of optical duplicates.
	`MergeBamAlignment`	Merge alignment data from a SAM or BAM with data in an unmapped BAM file.
	`SamToFastq`	Convert a SAM or BAM file to a FASTQ file.

Package (version)	Tool	Function as used in this document
Java (≥ 1.8.0_282)	`java`	Required for GATK4.
SAMtools (1.12)	`faidx`	Generate a FASTA index of the reference genome.
	`fastq`	Convert a BAM into FASTQ format.
	`flagstat`	Count the number of alignments for each FLAG type in a BAM file.
	`index`	Generate an index of the BAM file.
	`mpileup`	Generate text pileup output for the BAM file.
	`stats`	Produce comprehensive statistics from a BAM file.
	`sort`	Sort a BAM file.
	`view`	Select alignments based on the SAM FLAG value.
seqtk (1.3-r106)	`sample`	Randomly subsample FASTQ file(s).
fastp (0.20.1)	`fastp`	Trim raw reads for quality and sequenced primer/adapter.
fgbio (1.3.0)	`ExtractUmisFromBam`	Extract UMIs and store them in the RX tag of the BAM file.
	`GroupReadsByUmi`	Identify and group reads originating from the same source molecule.
	`CallDuplexConsensusReads`	Calculate the consensus sequence for each group of reads identified as originating from the same unique source molecule.
VarDict (1.8.2)	`vardict-java`	Call somatic variants from a BAM file.
	`teststrandbias.R`	Perform a statistical test to detect strand bias.
	`var2vcf_valid.pl`	Convert the variant output from the intermediate tabular file into a VCF file.
VarScan2 (2.4.4)	`varscan somatic`	Call somatic variants from BAM files of a matched tumor normal pair.
BEDTools (2.30.0)	`intersect`	Screen for overlaps between two sets of genomic features.
MSIsensor (0.5)	`scan`	Scan homopolymers and microsatellites.
	`msi`	Calculate msi score.
MSIsensor2 (0.1)	`msi`	Build machine learning models and calculate msi score.
CNVkit (0.9.8)	`access`	Calculate sequence accessible coordinates.
	`batch`	Run the CNVkit pipeline on BAM files.

Package (version)	Tool	Function as used in this document
STAR (2.7.8a)	`alignReads`	Map RNA sequencing reads to an indexed genome.
STAR-Fusion (1.10.0)	`STAR-Fusion`	Identify candidate fusion transcripts supported by reads.
CTAT-Splicing (0.0.2)	`CTAT-Splicing`	Identify exon-skipping and alternative splicing events.

Table 1: Third-party data analysis tools used in this white paper. The examples described in this document were tested using the software versions listed in parentheses, and different software versions may require different function calls and/or flags. See section *Reference Links* for installation instructions and explanations of command options. These tools were tested on a Redhat Linux system. *Many GATK4 tools were originally developed as part of Picard, which maintains detailed documentation referenced throughout this white paper.

Index a Reference Genome

Most Next-Generation Sequencing (NGS) mapping algorithms require an indexed genome to be created before mapping. Although algorithms work in different ways, most use the Burrows-Wheeler algorithm for mapping millions of relatively short reads against the reference genome. A genomic index is used to very quickly find the mapping location. Genomic indexing is required only once per genome version. The genomic index files that are created can then be used for all subsequent mapping jobs against that genome assembly version.

We recommend the FASTA formatted genome sequence be indexed with chromosomes in "karyotypic" sort order, *i.e.,* chr1, chr2, ..., chr10, chr11, … chrX, chrY, chrM, *etc.* In these examples, reference genome files are referred to as "ref.fa", which should be replaced by the actual file name (*e.g.,* "hg38.fa").

Package➔Tool(s) Used	BWA➔`index` SAMtools➔`faidx` GATK➔`CreateSequenceDictionary`
Input(s)	ref.fa
Output(s)	ref.fa {indexed} ref.fa = unmodified reference genome ref.fa.amb, ref.fa.ann, ref.fa.bwt, ref.fa.pac, ref.fa.sa = reference genome index files ref.fa.fai = FASTA index ref.dict = reference sequence dictionary

Generate Reference Genome Index

```
/path/to/bwa index -a bwtsw /path/to/ref.fa
```

Generate FASTA Index

```
/path/to/samtools faidx /path/to/ref.fa
```

Generate Sequence Dictionary

```
/path/to/gatk CreateSequenceDictionary --REFERENCE /path/to/ref.fa
```

The requirement for use of an indexed reference genome in a subsequent step is designated by "ref.fa {indexed}" in the Input(s) section. An indexed reference genome consists of the genome FASTA file and all index files present in the same directory.

Note that there can be multiple versions of the same reference genome available, and selecting the appropriate genome for your analyses is important. Ensure the reference genome build used to generate panel design files (such as primary target or capture target bed files) and used in the analysis pipeline are the same. Chromosome names should match in the two files. If the version of the reference genome contains extra chromosomes or contigs (i.e., ALT contigs in the human reference), consider if these are useful or necessary for your particular analysis. Some regions, such as the pseduo-autosomal regions in the human reference genome, can be represented in different ways that may affect downstream analyses including variant analysis.

Decompress a FASTQ File

If the FASTQ files have been compressed (with a .gz extension), some tools require them to be decompressed before use.

Package➔Tool(s) Used	`gunzip`
Input(s)	SAMPLE_R1.fastq.gz SAMPLE_R2.fastq.gz
Output(s)	SAMPLE_R1.fastq SAMPLE_R2.fastq

```
gunzip -c SAMPLE_R1.fastq.gz > SAMPLE_R1.fastq


gunzip -c SAMPLE_R2.fastq.gz > SAMPLE_R2.fastq
```

Examine Sequence Read Quality

Before spending time evaluating mapping statistics, use fastqc on raw reads and generate a per-base sequence quality plot and report to evaluate sequencing quality. The fastqc tool can work on both compressed and uncompressed FASTQ files.

Package➔Tool(s) Used	FastQC➔ `fastqc`
Input(s)	SAMPLE_R1.fastq / SAMPLE_R1.fastq.gz SAMPLE_R2.fastq / SAMPLE_R2.fastq.gz
Output(s)	SAMPLE_R1_fastqc.zip SAMPLE_R2_fastqc.zip

```
/path/to/fastqc --nogroup --extract SAMPLE_R1.fastq(.gz) SAMPLE_R2.fastq(.gz)
```

FastQC has a --threads option that allows users to specify the number of files which can be processed simultaneously. FastQC describes, "Each thread will be allocated 250MB of memory so you shouldn't run more threads than your available memory will cope with, and not more than 6 threads on a 32 bit machine". In the above example, users can specify --threads 2 to speed up the calculation. A .zip file is created for each SAMPLE input file. An HTML report named fastqc_report.html is created that is viewable in an internet browser. The authors of FastQC have posted the following examples of the QC report for a good and a bad sequencing run:
http://www.bioinformatics.babraham.ac.uk/projects/fastqc/good_sequence_short_fastqc.html
http://www.bioinformatics.babraham.ac.uk/projects/fastqc/bad_sequence_fastqc.html

Remove Duplicates by Utilizing Unique Molecular Identifiers (UMIs)

The recommended methodology to obtain consensus reads from PCR and optical duplicates are described in this section. It is specific for sequencing data from libraries constructed using the KAPA Universal UMI Adapter or the

AVENIO Edge HyperPlex UMI Adapter and the KAPA UDI Primer Mixes or the AVENIO Edge UDI Primer Mixes. The methodology illustrated below consists of three steps: 1) Extraction of the UMI from insert reads, 2) Grouping of these UMIs into families/groups based on alignment coordinates and UMI sequence composition, and 3) Consensus calling of all reads within a particular UMI group. The Roche UMI adapters described above are utilizing a mix of "fixed" sequence adapters. In case the exact UMI sequences are needed for additional analysis, please, refer to Table 2 below.

Note that the two steps "Perform Adapter Trimming and Quality Filtering" and "Map Reads to the Reference Genome" are integrated in this UMI deduplication process.

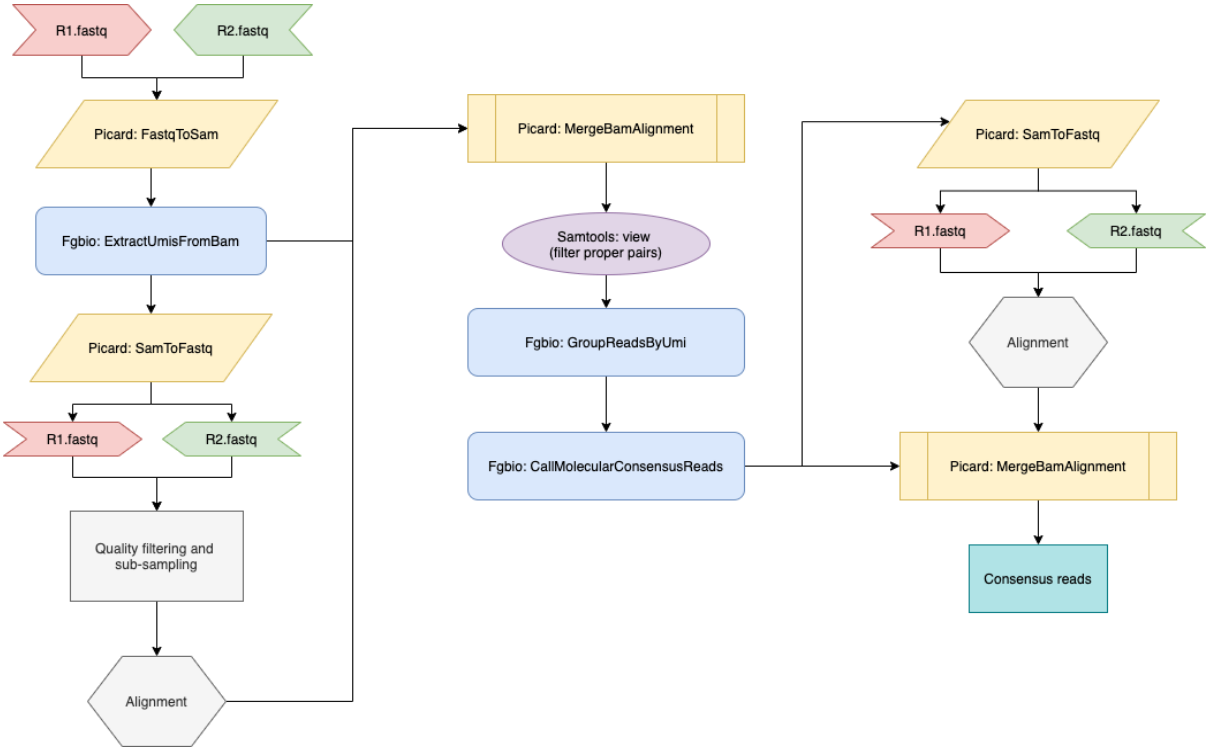

Figure 2: Recommended pipeline for UMI extraction, grouping and consensus read calling.

UMI family	Sequence	UMI family	Sequence
1	AATC**C**T	9	ACC**T**
2	AGAA**G**T	10	ATG**T**
3	CCAG**G**T	11	CAG**T**
4	CTTA**C**T	12	CGC**T**
5	GAAG**C**T	13	GCG**T**
6	GGTC**G**T	14	GTC**T**
7	TCTA**G**T	15	TAC**T**
8	TTAC**C**T	16	TGG**T**

Table 2: Sequence mix of the Roche UMI adapters (KAPA Universal UMI Adapter and AVENIO Edge HyperPlex UMI Adapter). The bolded T is the 3' T-overhang of the adapter. The sequence in italics is not part of the UMI sequence; it is added to increase the sequence diversity in order to ensure optimal sequencing performance.

Convert FASTQ to BAM

The first step is to convert the demultiplexed, raw sequencing FASTQ files to BAM files using the `FastqToSam` tool in GATK.

Package➔Tool(s) Used	GATK➔`FastqToSam`
Input(s)	SAMPLE_R1.fastq.gz SAMPLE_R2.fastq.gz
Output(s)	SAMPLE_unmapped.bam

```
/path/to/gatk FastqToSam \
  -F1 SAMPLE_R1.fastq.gz \
  -F2 SAMPLE_R2.fastq.gz \
  -O SAMPLE_unmapped.bam \
  -SM SAMPLE
```

Extract UMIs from BAM

The read structure is defined as 3M3S+T. Extract the first three bases and store them as the UMI in the RX tag of the BAM file (3M). Trim the subsequent three bases off the start of the read (3S). These bases constitute a punctuation sequence that increases the sequence diversity to ensure optimal sequencing performance. Maintain the remaining sequence as part of the insert read (+T). The UMIs extracted from read 1 and read 2 are stored in the RX tag of the unmapped BAM file as UMI1-UMI2 (hereafter referred to as "the UMI" and considered as a single sequence).

Package➔Tool(s) Used	fgbio➔`ExtractUmisFromBam`
Input(s)	SAMPLE_unmapped.bam
Output(s)	SAMPLE_unmapped_umi_extracted.bam

```
/path/to/fgbio ExtractUmisFromBam \
  -i SAMPLE_unmapped.bam \
  -o SAMPLE_unmapped_umi_extracted.bam \
  -r 3M3S+T 3M3S+T \
  -t RX \
  -a true
```

Perform Adapter Trimming and Quality Filtering

The BAM file with UMI extracted reads needs to be converted to a FASTQ file for adapter trimming and quality filtering. Adapter trimming and quality filtering should only take place after UMI extraction, to avoid any bias and ensure that only the template/insert is trimmed. In this workflow, the unmapped BAM file is first converted to FASTQ using GATK, and then adapter trimming and quality filtering are performed using fastp. Parameters are set so that the tool automatically detects adapter sequences or adapter sequences can be set (available in Illumina

Adapter Sequences Document #1000000002694 v.11 or later). NOTE: BAM to FASTQ conversion does not retain extracted UMI information. Thus, it is important to retain the output file from the UMI extraction, SAMPLE_unmapped_umi_extracted.bam, to preserve the UMI information stored in the RX tag that is used downstream after genomic alignment.

Package➔Tool(s) Used	GATK➔`SamToFastq` fastp
Input(s)	SAMPLE_unmapped_umi_extracted.bam
Output(s)	SAMPLE_umi_extracted_trimmed_R1.fastq SAMPLE_umi_extracted_trimmed_R2.fastq SAMPLE_fastp.log

Convert BAM to FASTQ

```
/path/to/gatk SamToFastq \
  -I SAMPLE_unmapped_umi_extracted.bam \
  -F SAMPLE_umi_extracted_R1.fastq \
  -F2 SAMPLE_umi_extracted_R2.fastq \
  --CLIPPING_ATTRIBUTE XT \
  --CLIPPING_ACTION 2
```

Perform Adapter and Quality Trimming

```
/path/to/fastp \
  -i SAMPLE_umi_extracted_R1.fastq \
  -o SAMPLE_umi_extracted_trimmed_R1.fastq \
  -I SAMPLE_umi_extracted_R2.fastq \
  -O SAMPLE_umi_extracted_trimmed_R2.fastq \
  -g -W 5 -q 20 -u 40 -x -3 -l 75 -c \
  -j fastp.json \
  -h fastp.html \
  -w NumProcessors &> SAMPLE_fastp.log
```

The `-3` and `-W 5` options allow trimming from the 3' tail in a sliding window of 5 bp. If the mean quality is below the quality set by `-q`, the bases are trimmed. In addition, `-u` specifies what percent of bases are allowed to be unqualified before a read is discarded. The `-x` and `-g` options turn on poly X and poly G tail trimming, respectively. The `-l` option means the length of the trimmed read must be at least 50 bp. The `-c` option turns on base correction for read pairs where read1 and read2 overlap.

The fastp application will produce two files. The SAMPLE_umi_extracted_trimmed_R1.fastq and SAMPLE_umi_extracted_trimmed_R2.fastq contain the reads that are still paired after adapter trimming and quality filtering. Unpaired reads can optionally be assigned to output files using the `--unpaired1` and `--unpaired2` options. If you want to increase the percentage of passing reads, the quality and length filters thresholds can be lowered.

Select a Subsample of Reads from a FASTQ File

Random subsampling is useful for normalizing the number of reads per set when doing comparisons. With paired end reads, it is important that the two files use the same values for the seed (`-s`) and number of reads. The seqtk application can write the sampled reads to uncompressed FASTQ files.

Package➔Tool(s) Used	seqtk➔`sample`
Input(s)	SAMPLE_umi_extracted_trimmed_R1.fastq SAMPLE_umi_extracted_trimmed_R2.fastq
Output(s)	SAMPLE_umi_extracted_trimmed_subset_R1.fastq SAMPLE_umi_extracted_trimmed_subset_R2.fastq

```
/path/to/seqtk sample -s 12345 SAMPLE_umi_extracted_trimmed_R1.fastq 30000000 > SAMPLE_umi_extracted_trimmed_subset_R1.fastq


/path/to/seqtk sample -s 12345 SAMPLE_umi_extracted_trimmed_R2.fastq 30000000 > SAMPLE_umi_extracted_trimmed_subset_R2.fastq
```

The commands above will randomly subsample 30 million matched read pairs from the paired end FASTQ files for a total of 60 million reads. Supplying the same random seed value (`-s`) ensures that the FASTQ records will remain in synchronized sort order and can be used for mapping, *etc*. Note that seqtk requires an amount of RAM proportional to the number of reads being subsampled. As you increase the size of the subsampled read set, more RAM is needed.

Map Reads to the Reference Genome

Adapter trimmed and quality filtered reads are mapped to the indexed reference genome using BWA. NOTE: RNA-Seq data will require a splice-aware aligner if aligning to a genome, e.g., STAR or HISAT2 aligner.

Package➔Tool(s) Used	BWA➔`mem`
Input(s)	SAMPLE_umi_extracted_trimmed_R1.fastq (shown below) or SAMPLE_umi_extracted_trimmed_subset_R1.fastq if subsampling SAMPLE_umi_extracted_trimmed_R2.fastq (shown below) or SAMPLE_umi_extracted_trimmed_subset_R2.fastq if subsampling ref.fa {indexed}
Output(s)	SAMPLE_umi_aligned.bam

```
/path/to/bwa mem \
  -R "@RG\tID:A\tDS:KAPA_TE\tPL:ILLUMINA\tLB:SAMPLE\tSM:SAMPLE" \
  -t NumProcessors -M \
  /path/to/ref.fa \
  SAMPLE_umi_extracted_trimmed_R1.fastq \
  SAMPLE_umi_extracted_trimmed_R2.fastq | \
  samtools view -Sb > SAMPLE_umi_aligned.bam
```

In the "Map Reads" step, the `-R` option defines the read group ("@RG"), which will appear in the BAM header. Within this string is the sample ID ("ID"), description field ("DS"), sequencing platform ("PL"), library name ("LB"), and sample name ("SM"). When a library name, ID and sample name do not separately exist, a SAMPLE descriptor may be used, as shown in the example above.

Add UMI Information to the Reads in BAM

As UMI information is not retained during BAM to FASTQ conversion, it is necessary to merge the two BAM files containing the UMI information (SAMPLE_unmapped_umi_extracted.bam) and the alignment coordinate information (SAMPLE_umi_aligned.bam). The UMI information is now stored in the RX tag of the new umi_extracted_aligned_merged.bam file.

Package➔Tool(s) Used	GATK➔`MergeBamAlignment`
Input(s)	SAMPLE_umi_aligned.bam SAMPLE_unmapped_umi_extracted.bam ref.fa {indexed}
Output(s)	SAMPLE_umi_extracted_aligned_merged.bam

```
/path/to/gatk MergeBamAlignment \
  --ATTRIBUTES_TO_RETAIN X0 \
  --ATTRIBUTES_TO_REMOVE NM \
  --ATTRIBUTES_TO_REMOVE MD \
  --ALIGNED_BAM SAMPLE_umi_aligned.bam \
  --UNMAPPED_BAM SAMPLE_unmapped_umi_extracted.bam \
  --OUTPUT SAMPLE_umi_extracted_aligned_merged.bam \
  --REFERENCE_SEQUENCE /path/to/ref.fa
  --SORT_ORDER queryname \
  --ALIGNED_READS_ONLY true \
  --MAX_INSERTIONS_OR_DELETIONS -1 \
  --PRIMARY_ALIGNMENT_STRATEGY MostDistant \
  --ALIGNER_PROPER_PAIR_FLAGS true \
  --CLIP_OVERLAPPING_READS false
```

Filter Reads

It is advisable to keep only reads that are aligned in proper pairs in BAM. The tool samtools `view` and the flag `-f2` can be used for this purpose.

Package➔Tool(s) Used	SAMtools➔`view`
Input(s)	SAMPLE_umi_extracted_aligned_merged.bam
Output(s)	SAMPLE_umi_extracted_aligned_merged_filtered.bam

```
/path/to/samtools view -f 2 -bh SAMPLE_umi_extracted_aligned_merged.bam > SAMPLE_umi_extracted_aligned_merged_filtered.bam
```

Identify and Group Reads Originating from the Same Source Molecule

The `GroupReadsByUmi` tool in fgbio utilizes the UMI (UMI1-UMI2) and the genomic alignment start site to assign unique source molecules to each applicable read. `GroupReadsByUmi` implements the adjacency strategy introduced by UMI-tools. The user can control how many errors/mismatches are allowed in the UMI sequence when assigning source molecules (`--edits=n`). UMI group statistics are output to a SAMPLE_umi_group_data.tsv file using the `-f` flag.

NOTE: The parameter `--edits=1` will account for a single mismatch in the entire UMI sequence (UMI1+UMI2). Altering this parameter to >1 will have a significant impact on the outcome of the UMI grouping algorithm and the resultant UMI groups.

Package➔Tool(s) Used	fgbio➔`GroupReadsByUmi`
Input(s)	SAMPLE_umi_extracted_aligned_merged_filtered.bam
Output(s)	SAMPLE_umi_grouped.bam SAMPLE_umi_group_data.tsv

```
/path/to/fgbio GroupReadsByUmi \
  --input=SAMPLE_umi_extracted_aligned_merged_filtered.bam \
  --output=SAMPLE_umi_grouped.bam \
  --strategy=paired \
  --edits=1 \
  -t RX \
  -f SAMPLE_umi_group_data.tsv
```

Calculate Consensus Sequence

The `CallDuplexConsensusReads` tool in fgbio processes each group of reads identified as originating from the same unique source molecule. The `–min-reads` flag defines the minimum number of reads required to form a consensus family. For example `–min-reads 1 0 0` requires at least one read from either strand, therefore the final consensus will include singletons. Users can modify the parameter based on needs, i.e. `–min-reads 3 1 1` requires at least 3 reads and at least 1 from each strand NOTE: Bases with a sequencing quality less than 20 will not be used in the consensus calculation but this can also be altered with the `–min-input-base-quality` flag.

NOTE: Here, reads are defined as those grouped into a UMI family/group, i.e., reads that have the same UMI tag and the same 5' start position.

Package➔Tool(s) Used	fgbio➔`CallDuplexConsensusReads`
Input(s)	SAMPLE_umi_grouped.bam
Output(s)	SAMPLE_umi_consensus_unmapped.bam

```
/path/to/fgbio CallDuplexConsensusReads \
  --input=SAMPLE_umi_grouped.bam \
  --output=SAMPLE_umi_consensus_unmapped.bam \
  --error-rate-post-umi 40 \
  --error-rate-pre-umi 45 \
  --min-reads 1 0 0 \
  --max-reads 50 \
  --min-input-base-quality 20 \
  --read-name-prefix='consensus'
```

Convert BAM to FASTQ

After consensus calling, the collapsing of the UMI groups results in the loss of alignment coordinate information. To rectify this, the SAMPLE_umi_consensus_unmapped.bam is converted to FASTQ format using `SamToFastq` in gatk.

Note: Loss of alignment coordinates is an inherent limitation of consensus calling and is related to alignment quality. When base information is statistically extrapolated from two or more molecules the alignment quality is also statistically averaged. Many downstream variant callers rely on alignment quality and thus, to avoid error, the consensus reads are realigned to ensure correct alignment qualities.

Package➔Tool(s) Used	GATK➔`SamToFastq`
Input(s)	SAMPLE_umi_consensus_unmapped.bam
Output(s)	SAMPLE_umi_consensus_unmapped_R1.fastq SAMPLE_umi_consensus_unmapped_R2.fastq

```
/path/to/gatk SamToFastq \
  -I SAMPLE_umi_consensus_unmapped.bam \
  -F SAMPLE_umi_consensus_unmapped_R1.fastq \
  -F2 SAMPLE_umi_consensus_unmapped_R2.fastq \
  --CLIPPING_ATTRIBUTE XT \
  --CLIPPING_ACTION 2
```

Map Consensus Reads to the Reference Genome

A new SAMPLE_umi_consensus_mapped.bam file is generated after aligning the consensus reads to the indexed reference genome using BWA.

Package➔Tool(s) Used	BWA➔`mem`
Input(s)	SAMPLE_umi_consensus_unmapped_R1.fastq SAMPLE_umi_consensus_unmapped_R2.fastq ref.fa {indexed}
Output(s)	SAMPLE_umi_consensus_mapped.bam

```
/path/to/bwa mem \
  -R "@RG\tID:A\tDS:KAPA_TE\tPL:ILLUMINA\tLB:SAMPLE\tSM:SAMPLE" \
  -v 3 -Y -M \
  -t NumProcessors \
  /path/to/ref.fa \
  SAMPLE_umi_consensus_unmapped_R1.fastq \
  SAMPLE_umi_consensus_unmapped_R2.fastq \
  | \
  samtools view -bh - > SAMPLE_umi_consensus_mapped.bam
```

Add UMI Information to the Consensus Reads in BAM

The final step is to merge the SAMPLE_umi_consensus_mapped.bam with the SAMPLE_umi_consensus_unmapped.bam to retain the UMI group information. This will yield an aligned BAM file with consensus reads and the UMI information retained in the RX flag.

Package➔Tool(s) Used	GATK➔`MergeBamAlignment`
Input(s)	SAMPLE_umi_consensus_mapped.bam SAMPLE_umi_consensus_unmapped.bam ref.fa {indexed}
Output(s)	SAMPLE_umi_deduped.bam

```
/path/to/gatk MergeBamAlignment \
  --ATTRIBUTES_TO_RETAIN X0 \
  --ATTRIBUTES_TO_RETAIN RX \
  --ALIGNED_BAM SAMPLE_umi_consensus_mapped.bam \
  --UNMAPPED_BAM SAMPLE_umi_consensus_unmapped.bam \
  --OUTPUT SAMPLE_umi_deduped.bam \
  --REFERENCE_SEQUENCE /path/to/ref.fa \
  --SORT_ORDER coordinate \
  --ADD_MATE_CIGAR true \
  --MAX_INSERTIONS_OR_DELETIONS -1 \
  --PRIMARY_ALIGNMENT_STRATEGY MostDistant \
  --ALIGNER_PROPER_PAIR_FLAGS true \
  --CLIP_OVERLAPPING_READS false
```

Sort BAM and Create Index

The BAM files need to be sorted and indexed for use in the subsequent steps. Here we sort and create indices for the BAM files both before and after the UMI deduplication. The BAM files can now be used for all downstream applications and analysis described by the particular NGS analysis workflow.

Package➔Tool(s) Used	SAMtools➔`sort` SAMtools➔`index`
Input(s)	SAMPLE_umi_aligned.bam SAMPLE_umi_deduped.bam
Output(s)	SAMPLE_umi_aligned_sorted.bam SAMPLE_umi_aligned_sorted.bam.bai SAMPLE_umi_deduped_sorted.bam SAMPLE_umi_deduped_sorted.bam.bai

Sort and Index the Non-deduped BAM

```
/path/to/samtools sort SAMPLE_umi_aligned.bam -@ NumProcessors -o
SAMPLE_umi_aligned_sorted.bam

/path/to/samtools index SAMPLE_umi_aligned_sorted.bam
```

Sort and Index the Deduped BAM

```
/path/to/samtools sort SAMPLE_umi_deduped.bam -@ NumProcessors -o
SAMPLE_umi_deduped_sorted.bam

/path/to/samtools index SAMPLE_umi_deduped_sorted.bam
```

Detect Somatic SNV and Indel

Tumor Only Mode

After reads are mapped and duplicates are removed, variants are often called against the reference genome. Somatic variants must be called by callers that are capable of detecting low abundance variants. Here we describe how to use VarDict to call somatic variants. VarDict has been a widely used somatic caller which is known as ultra sensitive to call variants from targeted sequencing data. VarDict's performance scales linearly to sequencing depth and it enables ultra-deep sequencing for tumor evolution and liquid biopsy. Originally written in Perl as `vardict.pl`, the tool has been developed with a java based replacement which yields 10 times acceleration than the Perl implementation.

Package→Tool(s) Used	`vardict-java`
	`teststrandbias.R`
	`var2vcf_valid.pl`

Input(s)	ref.fa {indexed} SAMPLE_umi_deduped_sorted.bam {indexed} DESIGN_capture_targets.bed
Output(s)	SAMPLE_vardict.vcf

Call Genomic Variants

```
/path/to/vardict-java \
  -G /path/to/ref.fa \
  -f AF_CUTOFF \
  -N SAMPLE \
  -b SAMPLE_umi_deduped_sorted.bam \
  -c 1 -S 2 -E 3 -g 4 DESIGN_capture_targets.bed \
  | \
  /path/to/teststrandbias.R \
  | \
  /path/to/var2vcf_valid.pl -N SAMPLE -E -f AF_CUTOFF > SAMPLE_vardict.vcf
```

The `-f` option specifies the threshold for allele fraction (default 0.01 or 1%). The allele fraction threshold should be adjusted for different applications. For example, 0.01 can be used for FFPE, while a lower value such as 0.0005 can be taken for cfDNA. The `-c`, `-s`, `-E` and `-g` options specify the columns for chromosome, region start, region end and gene annotation. Two other scripts are installed automatically together with `vardict-java`. The `teststrandbias.R` script performs a statistical test to detect strand bias. The `var2vcf_valid.pl` script converts the variant output from the intermediate tabular file into a validated VCF file. The `var2vcf_valid.pl -f` option sets the minimum allele fraction (default 0.02 or 2%) for filtering variants.

See *https://github.com/AstraZeneca-NGS/VarDictJava* and *https://github.com/AstraZeneca-NGS/VarDict* for details on VarDict usage.

Tumor-Normal Mode

VarScan2 can detect somatic mutations from tumor normal pairs. It reads data from the paired samples, and classifies variants by somatic status (germline, somatic or LOH) when a comparison of normalized read depth characterizes copy number changes. VarScan2 calls somatic variants using a heuristic method and a statistical test based on the number of aligned reads supporting each allele.

Package➔Tool(s) Used	SAMtools➔`mpileup` varscan➔`somatic`
Input(s)	ref.fa {indexed} SAMPLE_normal_umi_deduped_sorted.bam SAMPLE_tumor_umi_deduped_sorted.bam
Output(s)	SAMPLE.snp SAMPLE.indel

Generate Normal Pileup

```
/path/to/samtools mpileup -f /path/to/ref.fa SAMPLE_normal_umi_deduped_sorted.bam > SAMPLE_normal_umi_deduped_sorted.pileup
```

Generate Tumor Pileup

```
/path/to/samtools mpileup -f /path/to/ref.fa SAMPLE_tumor_umi_deduped_sorted.bam > SAMPLE_tumor_umi_deduped_sorted.pileup
```

Call Genomic Variants

```
/path/to/varscan somatic SAMPLE_normal_umi_deduped_sorted.pileup SAMPLE_tumor_umi_deduped_sorted.pileup SAMPLE --min-var-freq AF_CUTOFF
```

VarScan2 expects the pileup files for tumor and normal samples as input, which can be built by SAMtools. Note that the bam files need to be position sorted as described in previous steps. VarScan2 generates two output files, SAMPLE.snp and SAMPLE.indel consisting of SNVs and indels detected from the paired tumor normal pair. See *http://varscan.sourceforge.net/somatic-calling.html* for details on the output format.

Detect Microsatellite Instability (MSI)

Tumor Only Mode

MSIsensor2 is a tool that uses machine-learning models to detect MSI given tumor-only sequencing data. It is shown that it yields comparably high performance as MSIsensor, which takes matched tumor normal pairs as input. MSIsensor2 is able to detect MSI in multiple sample types including cfDNA and FFPE. In addition, it applies to different scales ranging from WGS, WES to targeted panel data. The tool works well for different targeted sequencing applications including amplicon sequencing.

The required inputs for MSIsensor2 tumor-only mode are a tumor bam file and a directory that stores models. Follow the instructions on *https://github.com/niu-lab/msisensor2* to install MSIsensor2 and get the models, which include models_hg38, models_hg19_GRCh37 and models_b37_HumanG1Kv37.

There are three output files generated by the `msi` module. The output.prefix file contains the MSI score. The output.prefix_dis file contains read count distribution for each site. The output.prefix_somatic file contains those somatic sites that are detected.

Package➔Tool(s) Used	MSIsensor2➔`msi`
Input(s)	SAMPLE_umi_deduped_sorted.bam models_hg38/
Output(s)	SAMPLE SAMPLE_dis SAMPLE_somatic
`/path/to/msisensor2 msi -M /path/to/models_hg38 -t SAMPLE_umi_deduped_sorted.bam -o SAMPLE -b NumProcessors`	

Tumor-Normal Mode

The original MSIsensor (Niu #) (Jia #) is used for deriving MSI status in the sequencing data of tumor normal pairs.

The `scan` step generates microsatellites.list that contains a list of microsatellites. The `msi` step generates four files: output.prefix, output.prefix_dis_tab, output.prefix_germline, and output.prefix_somatic. The MSI score is stored in the output.prefix file. See *https://github.com/ding-lab/msisensor/blob/master/README_msisensor.md#output* for a description of the output formats.

Package➔Tool(s) Used	MSIsensor➔`scan` MSIsensor➔`msi`
Input(s)	ref.fa {indexed} SAMPLE_normal_umi_deduped_sorted.bam SAMPLE_tumor_umi_deduped_sorted.bam
Output(s)	SAMPLE SAMPLE_dis SAMPLE_somatic SAMPLE_germline

Scan Homopolymers and Microsatellites

```
/path/to/msisensor scan -d /path/to/ref.fa -o microsatellites.list
```

Calculate MSI Score

```
/path/to/msisensor msi -d microsatellites.list -n
SAMPLE_normal_umi_deduped_sorted.bam -t SAMPLE_tumor_umi_deduped_sorted.bam -o
SAMPLE -b NumProcessors
```

Detect Copy Number Variation (CNV)

Tumor Only Mode

CNVkit (Talevich #) is a tool to identify copy numbers from high-throughput genome-wide sequencing data. See *https://cnvkit.readthedocs.io/en/stable/quickstart.html* for a description of the parameters and outputs.

The first step is to create a bed file that contains the locations of the accessible regions for a given reference genome. In the reference genome, the inaccessible regions including centromeres, telomeres and highly repetitive regions are avoided by CNVkit. If the user has other known unmappable or other regions that should be excluded, the `--exclude/-x` option can be used.

Package➔Tool(s) Used	CNVkit➔`access`
Input(s)	ref.fa {indexed}
Output(s)	accessible.bed

```
/path/to/cnvkit.py access /path/to/ref.fa -o accessible.bed
```

When there are a set of normals, they can serve as the reference or control samples for CNV calling in the tumor sample. Ideally the control samples should be of the same sample type, library preparation and platform as the tumor sample. The `batch` command in `cnvkit.py` automatically creates the pooled reference of per-bin copy number estimates from the normal samples and then uses the reference in processing the tumor sample. The `--fasta/-f`

option is needed to extract GC and RepeatMasker information for bias corrections. CNVkit leverages the information to improve the copy ratio estimates. The `--access/-g` option specifies regions of accessible sequence on chromosomes. It can be created by the `access` command in `cnvkit.py` as described in the above step. The `--annotate` option specifies the gene models for assigning names to the target regions. It can be a UCSC refFlat.txt or ensFlat.txt file, or BED, interval list, GFF, or similar. For example, the refFlat.txt.gz file for hg38 can be downloaded from *http://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/refFlat.txt.gz* and the refFlat.txt.gz file for hg19 can be downloaded from *http://hgdownload.soe.ucsc.edu/goldenPath/hg19/database/refFlat.txt.gz*. Note that the refFlat.txt.gz file needs to be decompressed by gunzip command as proper input of CNVkit.

Package➔Tool(s) Used	CNVkit➔`batch`
Input(s)	ref.fa {indexed} SAMPLE_tumor_umi_deduped_sorted.bam *normal_umi_deduped_sorted.bam (a batch of normals) DESIGN_capture_targets.bed accessible.bed refFlat.txt
Output(s)	SAMPLE_reference.cnn SAMPLE/ SAMPLE_tumor_umi_deduped_sorted.bintest.cns SAMPLE_tumor_umi_deduped_sorted.call.cns SAMPLE_tumor_umi_deduped_sorted.cnr SAMPLE_tumor_umi_deduped_sorted.cns SAMPLE_tumor_umi_deduped_sorted.antitargetcoverage.cnn SAMPLE_tumor_umi_deduped_sorted.targetcoverage.cnn *normal_umi_deduped_sorted.antitargetcoverage.cnn (for each normal in the batch) *normal_umi_deduped_sorted.targetcoverage.cnn (for each normal in the batch) DESIGN_capture_targets.antitarget.bed DESIGN_capture_targets.target.bed

```
/path/to/cnvkit.py batch \
  SAMPLE_tumor_umi_deduped_sorted.bam \
  -n *normal_umi_deduped_sorted.bam \
  -t DESIGN_capture_targets.bed \
  -f /path/to/ref.fa \
  --annotate refFlat.txt \
  --access accessible.bed \
  -p NumProcessors \
  --output-reference SAMPLE_reference.cnn \
  -d SAMPLE
```

When there are no normal samples available, a “flat” reference can be created which contains a neutral copy number for each probe. In this case, the `--normal/-n` option can be used without specifying any BAM files. Note, that when using the “flat” reference, the correction for the target capture bias is not possible, however the GC bias correction is still performed.

Package➔Tool(s) Used	CNVkit➔`batch`
Input(s)	ref.fa {indexed} SAMPLE_tumor_umi_deduped_sorted.bam DESIGN_capture_targets.bed accessible.bed refFlat.txt
Output(s)	SAMPLE_flat_reference.cnn SAMPLE/ SAMPLE_tumor_umi_deduped_sorted.bintest.cns SAMPLE_tumor_umi_deduped_sorted.call.cns SAMPLE_tumor_umi_deduped_sorted.cnr SAMPLE_tumor_umi_deduped_sorted.cns SAMPLE_tumor_umi_deduped_sorted.antitargetcoverage.cnn SAMPLE_tumor_umi_deduped_sorted.targetcoverage.cnn DESIGN_capture_targets.antitarget.bed DESIGN_capture_targets.target.bed

```
/path/to/cnvkit.py batch \
  SAMPLE_tumor_umi_deduped.bam \
  -n \
  -t DESIGN_capture_targets.bed \
  -f /path/to/ref.fa \
  --annotate refFlat.txt \
  --access accessible.bed \
  -p NumProcessors \
  --output-reference SAMPLE_flat_reference.cnn \
  -d SAMPLE
```

CNVkit outputs multiple tab-separated plain text files. The .cnn file stores bin-level coverages. The .cnr file stores bin-level log2 ratios with the "weight" column representing each bin's proportional weight or reliability. The .cns file stores segmented log2 ratios with the "probes" column representing the number of bins covered by the segment. CNVkit has a number of segmentation algorithms available, such as CBS, lasso, haar, hmm etc. The CBS algorithm is used by default, but it can be changed using the '--segment-method' option. See *https://cnvkit.readthedocs.io/en/stable/fileformats.html#file-formats* for detailed explanation of file formats.

Tumor-Normal Mode

CNVkit recommends combining a set of normal samples into a pooled reference even when matched tumor normal pairs are sequenced.

"To analyze a cohort sequenced on a single platform, we recommend combining all normal samples into a pooled reference, even if matched tumor-normal pairs were sequenced – our benchmarking showed that a pooled reference performed slightly better than constructing a separate reference for each matched tumor-normal pair. Furthermore, even matched normals from a cohort sequenced together can exhibit distinctly different copy number biases (see Plagnol et al. 2012 and Backenroth et al. 2014); reusing a pooled reference across the cohort provides some consistency to help diagnose such issues." (Reference: *https://cnvkit.readthedocs.io/en/stable/pipeline.html#paired-or-pooled-normals*)

Consequently, for matched tumor normal pairs, we suggest users combine multiple normals into a single reference pool and follow the instructions in the "Tumor Only" section for analysis. If it is still desired, the tumor-normal paired analysis can be run as below.

Package➔Tool(s) Used	CNVkit➔`batch`
Input(s)	ref.fa {indexed} SAMPLE_tumor_umi_deduped_sorted.bam SAMPLE_normal_umi_deduped_sorted.bam DESIGN_capture_targets.bed accessible.bed refFlat.txt
Output(s)	SAMPLE_reference.cnn SAMPLE/ SAMPLE_tumor_umi_deduped_sorted.bintest.cns SAMPLE_tumor_umi_deduped_sorted.call.cns SAMPLE_tumor_umi_deduped_sorted.cnr SAMPLE_tumor_umi_deduped_sorted.cns SAMPLE_tumor_umi_deduped_sorted.antitargetcoverage.cnn SAMPLE_tumor_umi_deduped_sorted.targetcoverage.cnn SAMPLE_normal_umi_deduped_sorted.antitargetcoverage.cnn SAMPLE_normal_umi_deduped_sorted.targetcoverage.cnn DESIGN_capture_targets.antitarget.bed DESIGN_capture_targets.target.bed

```
/path/to/cnvkit.py batch \
  SAMPLE_tumor_umi_deduped_sorted.bam \
  -n SAMPLE_normal_umi_deduped_sorted.bam \
  -t DESIGN_capture_targets.bed \
  -f /path/to/ref.fa \
  --annotate refFlat.txt \
  --access accessible.bed \
  -p NumProcessors \
  --output-reference SAMPLE_reference.cnn \
  -d SAMPLE
```

Detect fusions and exon skipping from RNA data *(not applicable for the AVENIO Edge System)*

Perform Adapter Trimming and Quality Filtering

Fastp is used to trim off some bases from 5' and 3' ends of reads in the FASTQ files. The `-f` and `-F` options allow trimming of 3 bases from 5' of both read pairs. The `-3` and `-W 5` options allow trimming from the 3' tail in a sliding window of 5 bp. The `-x` and `-g` options turn on poly X and poly G tail trimming, respectively. If the mean quality is below the quality set by `-q` (default 15), the bases are trimmed. In addition, `-u` (default 40) specifies what percent of bases are allowed to be unqualified before a read is discarded. The `-l` (default 15) option means the minimum length of the trimmed read to be retained.

Package➔Tool(s) Used	fastp
Input(s)	SAMPLE_R1.fastq SAMPLE_R2.fastq
Output(s)	SAMPLE_R1_trimmed.fastq SAMPLE_R2_trimmed.fastq SAMPLE_trimmed.json SAMPLE_trimmed.html SAMPLE_fastp.log

```
/path/to/fastp \
  -i SAMPLE_R1.fastq \
  -o SAMPLE_R1_trimmed.fastq \
  -I SAMPLE_R2.fastq \
  -O SAMPLE_R2_trimmed.fastq \
  -f 3 -F 3 -3 -W 5 -q 20 -u 40 -l 50 -x -g \
  -j SAMPLE_trimmed.json \
  -h SAMPLE_trimmed.html \
  -w NumProcessors &> SAMPLE_fastp.log
```

Map Reads to the rRNA Reference Genome

Next reads are aligned to rRNA reference using BWA `mem`. The reads originating from rRNA need to be further removed for more accurate quantification of reads supporting fusions. Note the rrna_ref.fa needs to be indexed as described in the "Index a Reference Genome" step. When counting the number of rRNA reads, `-F2828` flag is used to remove reads with following attributes: read unmapped; mate unmapped; not primary alignment; read fails platform/vendor quality checks; supplementary alignment.

Package➔Tool(s) Used	BWA➔mem SAMtools⇨view
Input(s)	rrna_ref.fa {indexed} SAMPLE_R1_trimmed.fastq SAMPLE_R2_trimmed.fastq
Output(s)	SAMPLE_rrna.bam SAMPLE_rRNA_read_counts.tsv

Map reads to rRNA reference and convert to BAM

```
/path/to/bwa mem \
  -R '@RG\tID:A\tSM:KAPA' \
  /path/to/rrna_ref.fa \
  -t NumProcessors \
  SAMPLE_R1_trimmed.fastq \
  SAMPLE_R2_trimmed.fastq | \

  /path/to/samtools view -u - > SAMPLE_rrna.bam
```

Count the Number and Percentage of rRNA Reads

```
RRNA_READ_CNT=$(/path/to/samtools view -F2828 -c SAMPLE_rrna.bam)
TOTAL_READ_LINE_CNT=$(wc -l SAMPLE_R1_trimmed.fastq | gawk -F " " '{print $1}')
TOTAL_READ_CNT=$((TOTAL_READ_LINE_CNT/2))
RRNA_RATE=$((100*RRNA_READ_CNT/TOTAL_READ_CNT))
echo -e "Number of rRNA reads\t$RRNA_READ_CNT" > SAMPLE_rRNA_read_counts.tsv
echo -e "% rRNA reads\t$RRNA_RATE" >> SAMPLE_rRNA_read_counts.tsv
```

Extract Unmapped Non-rRNA Reads

The unmapped reads from the previous step represent those that are of non-rRNA origin and are next extracted for mapping to the human genome. The `-f 0xC` flag is set to select reads and mates that are unmapped.
The `-F 0x900` flag is set to exclude non-primary or supplementary alignments.

Package➔Tool(s) Used	SAMtools⇨fastq
Input(s)	SAMPLE_rrna.bam
Output(s)	SAMPLE_rrna_unmapped_R1.fastq SAMPLE_rrna_unmapped_R2.fastq SAMPLE_rrna_unmapped_unpaired.fastq SAMPLE_rRNA_read_counts.tsv

Convert BAM to FASTQ

```
/path/to/samtools fastq \
  -f 0xC -F 0x900 \
  -1 SAMPLE_rrna_unmapped_R1.fastq \
  -2 SAMPLE_rrna_unmapped_R2.fastq \
  -s SAMPLE_rrna_unmapped_unpaired.fastq \
  SAMPLE_rrna.bam
```

Count the Number of Reads after rRNA Removal

```
READ_LINE_CNT=$(wc -l SAMPLE_rrna_unmapped_R1.fastq | gawk -F " " '{print $1}')
READ_CNT=$((READ_LINE_CNT/2))
echo -e "Number of reads after rRNA removal\t$READ_CNT" >> SAMPLE_rRNA_read_counts.tsv
```

Prepare Genome Indices Files

STAR-Fusion is used to identify candidate fusion transcripts supported by junction reads and spanning reads. There are two ways to run STAR-Fusion. The typical way starts directly from the FASTQ files. The other way requires running STAR first to generate a "Chimeric.junction.out" file, which will be further leveraged by STAR-Fusion. Here we describe how to run STAR and STAR-Fusion sequentially. This mode offers more flexibility to users when STAR has already been run earlier on, or STAR is preferred to run separately to use the outputs in other processes such as for expression estimates or variant detection. See *https://github.com/STAR-Fusion/STAR-Fusion/wiki* for more information on the two modes.

To start off, both STAR and STAR-Fusion require genome libraries. Users can download the libraries curated by the Trinity Cancer Transcriptome Analysis Toolkit (CTAT). Data resources required are readily available including the human genome, Gencode annotations, and coding annotations in gtf format. Besides, precomputed BLAST+ results, Pfam domains identified in human protein sequences, and human cancer fusion annotations are also included. Note that CTAT genome libraries are specific for corresponding STAR and STAR-Fusion software releases. Users can find the STAR/STAR-Fusion release and CTAT Genome Library Compatibility Matrix on this page *https://github.com/STAR-Fusion/STAR-Fusion/wiki/STAR-Fusion-release-and-CTAT-Genome-Lib-Compatibility-Matrix*.

As an example, when STAR v2.7.8a and STAR-Fusion v1.10.0 are installed, the corresponding CTAT genome library *CTAT genome lib StarFv1.10* should be downloaded. On the resource page, there are two kinds of genome libraries with the key word "plug-n-play" or "source". The "plug-n-play" file is a pre-compiled CTAT genome library. The download is larger and takes longer, but it includes all processed data and saves users from having to run through the time-intensive and computationally-intensive build process. In comparison, the "source" file requires executing the genome library build process manually. See *https://github.com/NCIP/ctat-genome-lib-builder/wiki* for information on building the genome libraries. Here we describe using the "plug-n-play" resources as genome libraries for STAR and STAR-Fusion. After downloading the pre-compiled plug-n-play.tar.gz file, users just need to unpack it using the `tar -xvzf` command. The path to ctat_genome_lib_build_dir folder can be provided for the `--genome_lib_dir` option in STAR-Fusion. The path to the nested folder ref_genome.fa.star.idx can be provided for the `--genomeDir` option in STAR.

Map Reads to the Reference Genome

The unmapped non-rRNA reads are mapped to the human genome. The folder ref_genome.fa.star.idx stores reference files used by STAR. See *https://github.com/alexdobin/STAR/blob/master/doc/STARmanual.pdf* for a description of the parameters and output files. The SAMPLE_star_Log.final.out file contains multiple metrics related to STAR alignment, such as "Uniquely mapped reads %". The output file SAMPLE_star_Chimeric.out.junction will be leveraged by STAR-Fusion in the next step.

Package➔Tool(s) Used	STAR⇨alignReads SAMtools⇨index
Input(s)	SAMPLE_rrna_unmapped_R1.fastq SAMPLE_rrna_unmapped_R2.fastq ref_genome.fa.star.idx/

	SAMPLE_star_Aligned.sortedByCoord.out.bam SAMPLE_star_Aligned.sortedByCoord.out.bam.bai SAMPLE_star_Chimeric.out.junction SAMPLE_star_unmapped_R1.fastq SAMPLE_star_unmapped_R2.fastq SAMPLE_star_Log.final.out
Output(s)	SAMPLE_human_read_counts.tsv

Map Reads to Human Genome

```
/path/to/STAR --runMode alignReads \
  --genomeDir /path/to/ref_genome.fa.star.idx \
  --readFilesIn SAMPLE_rrna_unmapped_R1.fastq SAMPLE_rrna_unmapped_R2.fastq \
  --runThreadN NumProcessors \
  --outSAMtype BAM SortedByCoordinate \
  --chimOutJunctionFormat 1 \
  --alignSJstitchMismatchNmax 5 -1 5 5 \
  --twopassMode Basic \
  --outReadsUnmapped Fastx \
  --chimSegmentMin 12 --chimJunctionOverhangMin 12 \
  --alignSJDBoverhangMin 10 --chimSegmentReadGapMax 3 \
  --outSAMstrandField intronMotif \
  --outFilterScoreMinOverLread 0.5 --outFilterMatchNminOverLread 0.5 \
  --outFileNamePrefix SAMPLE_star_
```

Create Index

```
/path/to/samtools index SAMPLE_star_Aligned.sortedByCoord.out.bam
```

Rename the FASTQ Files

```
mv SAMPLE_star_Unmapped.out.mate1 SAMPLE_star_unmapped_R1.fastq

mv SAMPLE_star_Unmapped.out.mate2 SAMPLE_star_unmapped_R2.fastq
```

Count the Number of Reads

```
READ_CNT=$(/path/to/samtools view -F2828 -c
SAMPLE_star_Aligned.sortedByCoord.out.bam)

echo -e "Number of reads mapped to human genome\t$READ_CNT" >
SAMPLE_human_read_counts.tsv
```

Identify Candidate Fusion Transcripts

STAR-Fusion is used to identify candidate fusion transcripts supported by reads. STAR-Fusion further processes the output generated by the STAR aligner to map junction reads and spanning reads to a reference annotation set. The ctat_genome_lib_build_dir represents the directory containing the genome library as described earlier.

Package→Tool(s) Used	STAR-Fusion
Input(s)	SAMPLE_star_Chimeric.out.junction ctat_genome_lib_build_dir/
Output(s)	SAMPLE_star_Chimeric.out/ star-fusion.fusion_predictions.tsv star-fusion.fusion_predictions.abridged.tsv

```
/path/to/STAR-Fusion \
  --genome_lib_dir /path/to/ctat_genome_lib_build_dir \
  -J SAMPLE_star_Chimeric.out.junction \
  --output_dir SAMPLE_star_Chimeric.out
```

The output from STAR-Fusion are star-fusion.fusion_predictions.tsv and star-fusion.fusion_predictions.abridged.tsv in the output folder. They are tab-delimited and the latter is an abridged version that excludes the identities of the fusion reads. See *https://github.com/STAR-Fusion/STAR-Fusion/wiki* for a description of output formats and more information on the recommended parameters for running STAR and STAR-Fusion.

Identify Exon Skipping and Aberrant Splicing Isoforms

CTAT-Splicing is used to identify aberrant splicing isoforms that may result from exon skipping or alternative splicing. CTAT-Splicing uses the output generated by the STAR aligner to map the splice junctions to a reference annotation set.

CTAT can be obtained from the CTAT-Splicing Releases area: https://github.com/NCIP/CTAT-SPLICING/releases. Docker (https://hub.docker.com/r/trinityctat/ctat_splicing) and Singularity (https://data.broadinstitute.org/Trinity/CTAT_SINGULARITY/CTAT-SPLICING/) images are also available.

CTAT-Splicing is compatible with the CTAT genome libraries distributed for use with STAR fusion as described above. Once CTAT genome lib is installed, CTAT-splicing data resource supplement can be integrated. Download the GRCh37 or GRCh38 'cancer_introns.*.tsv.gz' file that matches the corresponding CTAT genome library being used. The 'cancer_introns.*.tsv.gz' can be integrated into the CTAT genome lib using the scripts provided by the CTAT-Splicing software:

```
/path/to/CTAT_SPLICING/prep_genome_lib/ctat-splicing-lib-integration.py \
            --cancer_introns_tsv cancer_introns.*.tsv.gz \
            --genome_lib_dir /path/to/CTAT_genome_lib_build_dir
```

Package⇨Tool(s) Used	CTAT-Splicing
Input(s)	SAMPLE_star_SJ.out.tab SAMPLE_star_Chimeric.out.junction (optional) SAMPLE_star_Aligned.sortedByCoord.out.bam (optional, required for visualization)
Output(s)	SAMPLE_star.introns SAMPLE_star.cancer.introns SAMPLE_star.cancer.introns.prelim SAMPLE_star.ctat-splicing.igv.html

```
/path/to/CTAT_SPLICING/STAR_to_cancer_introns.py \
  --SJ_tab_file SAMPLE_star_SJ.out.tab \
  --chimJ_file SAMPLE_star_Chimeric.out.junction \
  --vis \
  --bam_file SAMPLE_star_Aligned.sortedByCoord.out.bam \
  --output_prefix SAMPLE \
  --sample_name SAMPLE
```

The output file star.cancer.introns contains the list of candidate 'cancer introns' which were found to be enriched in cancer transcriptome samples. See *https://github.com/NCIP/CTAT-SPLICING/wiki* for description of output formats and more information.

Calculate On-Target Rates

Several metrics can be calculated such as on-target rates of reads mapped to housekeeping and fusion genes in design.

Package➔Tool(s) Used	BEDTools➔`intersect`
Input(s)	SAMPLE_star_Aligned.sortedByCoord.out.bam DESIGN_housekeeping_genes.bed DESIGN_capture_targets.bed
Output(s)	SAMPLE_housekeeping_ontarget.tsv

Count total reads

```
TOTAL_READ_CNT=$(samtools flagstat SAMPLE_star_Aligned.sortedByCoord.out.bam | grep
"mapped (" | gawk -F " " '{print $1}')
```

Count total on-target reads

```
ONTARGET_READ_CNT=$(bedtools intersect -bed -a
SAMPLE_star_Aligned.sortedByCoord.out.bam -b DESIGN_capture_targets.bed | cut -f 4 |
sort | uniq | wc -l | gawk -F " " '{print $1}')

ONTARGET_RATE=$((100*ONTARGET_READ_CNT/TOTAL_READ_CNT))

echo -e "% Reads on-target for entire panel\t$ONTARGET_RATE" >
SAMPLE_on_target_rate.tsv
```

Count housekeeping on-target reads

```
HOUSEKEEPING_ONTARGET_READ_CNT=$(bedtools intersect -bed -a
SAMPLE_star_Aligned.sortedByCoord.out.bam -b DESIGN_housekeeping_genes.bed | cut -f 4
| sort | uniq | wc -l | gawk -F " " '{print $1}')

HOUSEKEEPING_ONTARGET_RATE=$((100*HOUSEKEEPING_ONTARGET_READ_CNT/TOTAL_READ_CNT))

echo -e "% Reads on-target housekeeping genes\t$HOUSEKEEPING_ONTARGET_RATE" >>
SAMPLE_on_target_rate.tsv
```

Count fusion on-target reads

```
FUSION_ONTARGET_READ_CNT=$((ONTARGET_READ_CNT - HOUSEKEEPING_ONTARGET_READ_CNT))

FUSION_ONTARGET_RATE=$((100*FUSION_ONTARGET_READ_CNT/TOTAL_READ_CNT))

echo -e "% Reads on-target fusion genes in design\t$FUSION_ONTARGET_RATE" >>
SAMPLE_on_target_rate.tsv
```

Basic Mapping Metrics

Basic mapping metrics can be calculated using GATK `CollectAlignmentSummaryMetrics`.

Package➔Tool(s) Used	GATK➔`CollectAlignmentSummaryMetrics`
Input(s)	ŕref.fa {indexed} SAMPLE_umi_aligned_sorted.bam SAMPLE_umi_deduped_sorted.bam SAMPLE_sorted_rmdups_gatk.bam
Output(s)	SAMPLE_alignment_metrics_sorted.txt SAMPLE_alignment_metrics_sorted_rmdups.txt SAMPLE_alignment_metrics_sorted_rmdups_gatk.txt

CollectAlignmentSummaryMetrics Before Duplicates Removal

```
/path/to/gatk CollectAlignmentSummaryMetrics \
  --METRIC_ACCUMULATION_LEVEL ALL_READS \
  --INPUT SAMPLE_umi_aligned_sorted.bam \
  --OUTPUT SAMPLE_alignment_metrics_sorted.txt \
  --REFERENCE_SEQUENCE /path/to/ref.fa \
  --VALIDATION_STRINGENCY LENIENT
```

CollectAlignmentSummaryMetrics After Position-based Duplicates Removal

```
/path/to/gatk CollectAlignmentSummaryMetrics \
  --METRIC_ACCUMULATION_LEVEL ALL_READS \
  --INPUT SAMPLE_sorted_rmdups_gatk.bam \
  --OUTPUT SAMPLE_alignment_metrics_sorted_rmdups_gatk.txt \
  --REFERENCE_SEQUENCE /path/to/ref.fa \
  --VALIDATION_STRINGENCY LENIENT
```

CollectAlignmentSummaryMetrics After UMI-based Duplicates Removal

```
/path/to/gatk CollectAlignmentSummaryMetrics \
  --METRIC_ACCUMULATION_LEVEL ALL_READS \
  --INPUT SAMPLE_umi_deduped_sorted.bam \
  --OUTPUT SAMPLE_alignment_metrics_sorted_rmdups.txt \
  --REFERENCE_SEQUENCE /path/to/ref.fa \
  --VALIDATION_STRINGENCY LENIENT
```

See *https://broadinstitute.github.io/picard/picard-metric-definitions.html#AlignmentSummaryMetrics* for a description of the output metrics.

Calculate Mapping Rate and Error Rate

Mapping rates ("% reads mapped" and "% paired reads mapped") can be calculated using SAMtools `flagstat`. Error rate (ratio between mismatches and bases mapped) can be calculated using SAMtools `stats`.

Package➔Tool(s) Used	SAMtools➔`flagstat` SAMtools➔`stats`
Input(s)	SAMPLE_umi_aligned_sorted.bam
Output(s)	SAMPLE_flagstat.txt SAMPLE_stats.txt

Calculate Mapping Rate
`/path/to/samtools flagstat SAMPLE_umi_aligned_sorted.bam > SAMPLE_flagstat.txt`
 	
Calculate Error Rate
`/path/to/samtools stats SAMPLE_umi_aligned_sorted.bam > SAMPLE_stats.txt`	

See *http://www.htslib.org/doc/samtools-flagstat.html* and *http://www.htslib.org/doc/samtools-stats.html* for a description of the output metrics.

Count Optical Duplicates

GATK `MarkDuplicates` is used to count the number of optical duplicates.

Package➔Tool(s) Used	GATK➔`MarkDuplicates`
Input(s)	SAMPLE_umi_aligned_sorted.bam
Output(s)	SAMPLE_sorted_rmdups_gatk.bam SAMPLE_sorted_rmdups_gatk.bai SAMPLE_markduplicates_metrics_gatk.txt

Mark Duplicates

```
/path/to/gatk MarkDuplicates \
  --VALIDATION_STRINGENCY LENIENT \
  -I SAMPLE_umi_aligned_sorted.bam \
  -O SAMPLE_sorted_rmdups_gatk.bam \
  --METRICS_FILE SAMPLE_markduplicates_metrics_gatk.txt \
  --REMOVE_DUPLICATES true \
  --ASSUME_SORTED true \
  --CREATE_INDEX true
```

View the file SAMPLE_markduplicates_metrics_gatk.txt for counts of paired, unpaired, and duplicate reads. See *https://broadinstitute.github.io/picard/picard-metric-definitions.html#DuplicationMetrics* for a description of the output metrics. Note that "optical duplicates" are also reported, based on sequence similarity and sequencing cluster distance. Optical duplicates are a subset of the total duplicate rate and are counted within the paired and unpaired duplicates. For patterned flow cells (e.g., HiSeq X and HiSeq 4000), `--OPTICAL_DUPLICATE_PIXEL_DISTANCE` should be changed from the default of 100 to 2500.

Estimate Insert Size Distribution

The DNA that goes into sequence capture is generated by random fragmentation and later size selected. It is normal to observe a range of fragment sizes, but if skewed too large or too small, the on-target rate and/or percent of bases covered with at least one read can be adversely affected. The size of these fragments can be estimated from paired end sequencing reads (will not work for single end reads).

Package➔Tool(s) Used	GATK➔`CollectInsertSizeMetrics`
Input(s)	SAMPLE_umi_aligned_sorted.bam SAMPLE_umi_deduped_sorted.bam
Output(s)	SAMPLE_insert_size_metrics_sorted.txt SAMPLE_insert_size_plot_sorted.pdf SAMPLE_insert_size_metrics_sorted_rmdups.txt SAMPLE_insert_size_plot_sorted_rmdups.pdf

Estimate Insert Size Before Duplicates Removal

```
/path/to/gatk CollectInsertSizeMetrics \
  --VALIDATION_STRINGENCY LENIENT \
  -H SAMPLE_insert_size_plot_sorted.pdf \
  -I SAMPLE_umi_aligned_sorted.bam \
  -O SAMPLE_insert_size_metrics_sorted.txt
```

Estimate Insert Size After Duplicates Removal

```
/path/to/gatk CollectInsertSizeMetrics \
  --VALIDATION_STRINGENCY LENIENT \
  -H SAMPLE_insert_size_plot_sorted_rmdups.pdf \
  -I SAMPLE_umi_deduped_sorted.bam \
  -O SAMPLE_insert_size_metrics_sorted_rmdups.txt
```

See *https://broadinstitute.github.io/picard/picard-metric-definitions.html#InsertSizeMetrics* for a description of output metrics included in SAMPLE_insert_size_metrics_sorted.txt for all reads and SAMPLE_insert_size_metrics_sorted_rmdups.txt for non-duplicate reads which can also be used to plot the insert size distributions across samples. As long as R is installed on your system, a PDF plot is also created.

Count On-Target Reads

Use GATK `CountReads` to calculate the number of reads that overlap a target BED file by at least 1 bp. Calculation of on-target reads is one measure of the success of a Roche TE experiment, though optimal on-target is design-specific. The on-target metric is affected by library insert size, hybridization and wash stringency, and laboratory protocol.

Package➔Tool(s) Used	GATK➔`CountReads`
Input(s)	ref.fa {indexed} SAMPLE_umi_aligned_sorted.bam SAMPLE_umi_deduped_sorted.bam DESIGN_capture_targets.bed
Output(s)	ontarget_reads_sorted.txt ontarget_reads_sorted_rmdups.txt

Count Reads Before Duplicates Removal

```
/path/to/gatk CountReads \
  -R /path/to/ref.fa \
  -I SAMPLE_umi_aligned_sorted.bam \
  -L DESIGN_capture_targets.bed \
  --read-filter MappedReadFilter \
  --read-filter NotSecondaryAlignmentReadFilter > ontarget_reads_sorted.txt
```

Count Reads After Duplicates Removal

```
/path/to/gatk CountReads \
  -R /path/to/ref.fa \
  -I SAMPLE_umi_deduped_sorted.bam \
  -L DESIGN_capture_targets.bed \
  --read-filter MappedReadFilter \
  --read-filter NotSecondaryAlignmentReadFilter > ontarget_reads_sorted_rmdups.txt
```

It is necessary to apply filters to include specific reads for analysis. In the example commands above, "MappedReadFilter" and "NotSecondaryAlignmentReadFilter" are used to filter out reads that are unmapped or representing secondary alignments. See https://gatk.broadinstitute.org/hc/en-us/articles/360057438571--Tool-Documentation-Index#ReadFilters for a full list of available read filters. Divide "the number of on-target reads" found in ontarget_reads_sorted_rmdups.txt by "the total number of mapped, non-duplicate reads" to get "the percentage of on-target reads after duplicates removal". Similarly, divide "the number of on-target reads" found in ontarget_reads_sorted.txt by "the total number of mapped reads" to get "the percentage of on-target reads before duplicates removal". See *Basic Mapping Metrics* for reporting of the total number of mapped and non-duplicate reads.

Target-adjacent coverage is typical for target enrichment due to the capture of partially on-target DNA library fragments that also extend outside the capture region. To optionally assess the amount of reads that are target

adjacent, add `--interval_padding 100` to the commands above to add 100 bp to both sides of all targets. Although 100 bp is commonly used for this kind of padding, shorter or longer lengths may also be appropriate depending on expected library fragment sizes. Please note: all remaining steps in this document are written to use non-padded targets.

Create Genomic Interval Lists

Interval lists are genomic interval description files required by GATK `CollectHsMetrics` that contain a SAM-like header describing the reference genome and a set of coordinates with strand and name for each interval. The Roche-provided "primary target" files can be provided as GATK "target interval" inputs, and the Roche-provided "capture target" files can be provided as GATK "bait interval" inputs. However, in the KAPA HyperPETE application, we focus on evaluating the capture performance on "capture target", and we want to get the `--PER_BASE_COVERAGE` and `--PER_TARGET_COVERAGE` options in GATK `CollectHsMetrics` applied to "capture target" for detailed outputs of the coverage in each base and each target region. Here we create the interval list for the "capture target" file, which will be provided as both the "target interval" input and the "bait interval" input in GATK `CollectHsMetrics`.

Use the GATK `BedToIntervalList` command to create Interval List files from target BED files.

Package➔Tool(s) Used	GATK➔`BedToIntervalList`
Input(s)	DESIGN_capture_targets.bed ref.dict {one of the files in the indexed genome file set}
Output(s)	DESIGN_bait.interval_list

Create a Genomic Bait Interval List

```
/path/to/gatk BedToIntervalList \
  --INPUT DESIGN_capture_targets.bed \
  --SEQUENCE_DICTIONARY /path/to/ref.dict \
  --OUTPUT DESIGN_bait.interval_list
```

The GATK `IntervalListTool` command (not described here) can be used to add padding to interval lists.

Hybrid Selection (HS) Analysis Metrics

The `CollectHsMetrics` command calculates a number of metrics assessing the quality of target enrichment reads.

Package➔Tool(s) Used	GATK➔`CollectHsMetrics`
Input(s)	ref.fa {indexed} SAMPLE_umi_aligned_sorted.bam {indexed} SAMPLE_umi_deduped_sorted.bam {indexed} DESIGN_bait.interval_list
Output(s)	SAMPLE_hs_metrics_sorted.txt SAMPLE_hs_metrics_sorted_rmdups.txt SAMPLE_per_base_coverage_sorted.txt SAMPLE_per_base_coverage_sorted_rmdups.txt SAMPLE_per_target_coverage_sorted.txt SAMPLE_per_target_coverage_sorted_rmdups.txt

CollectHsMetric Before Duplicates Removal

```
/path/to/gatk CollectHsMetrics \
  --BAIT_INTERVALS DESIGN_bait.interval_list \
  --BAIT_SET_NAME DESIGN \
  --TARGET_INTERVALS DESIGN_bait.interval_list \
  --INPUT SAMPLE_umi_aligned_sorted.bam \
  --OUTPUT SAMPLE_hs_metrics_sorted.txt \
  --METRIC_ACCUMULATION_LEVEL ALL_READS \
  --REFERENCE_SEQUENCE /path/to/ref.fa \
  --VALIDATION_STRINGENCY LENIENT \
  --COVERAGE_CAP 100000 \
  --PER_BASE_COVERAGE SAMPLE_per_base_coverage_sorted.txt \
  --PER_TARGET_COVERAGE SAMPLE_per_target_coverage_sorted.txt
```

CollectHsMetric After Duplicates Removal

```
/path/to/gatk CollectHsMetrics \
  --BAIT_INTERVALS DESIGN_bait.interval_list \
  --BAIT_SET_NAME DESIGN \
  --TARGET_INTERVALS DESIGN_bait.interval_list \
  --INPUT SAMPLE_umi_deduped_sorted.bam \
  --OUTPUT SAMPLE_hs_metrics_sorted_rmdups.txt \
  --METRIC_ACCUMULATION_LEVEL ALL_READS \
  --REFERENCE_SEQUENCE /path/to/ref.fa \
  --VALIDATION_STRINGENCY LENIENT \
  --COVERAGE_CAP 100000 \
  --PER_BASE_COVERAGE SAMPLE_per_base_coverage_sorted_rmdups.txt \
  --PER_TARGET_COVERAGE SAMPLE_per_target_coverage_sorted_rmdups.txt
```

Additional Levels of Coverage (see note below for calculation)

```
gawk '$1 != "chrom" && $4 >= N' SAMPLE_per_base_coverage_sorted_rmdups.txt | wc -l
```

Here we supply the same interval file to both `--TARGET_INTERVALS` and `--BAIT_INTERVALS` parameters of GATK `CollectHsMetrics`, as we want to leverage the `--PER_BASE_COVERAGE` and `--PER_TARGET_COVERAGE` options to examine the per base and per target coverages in those capture regions. See *https://broadinstitute.github.io/picard/picard-metric-definitions.html#HsMetrics* for a description of output metrics. Note that some metrics are not directly comparable as some are obtained before read or base filters are applied (e.g., capture bases metrics) while others are calculated after (e.g., target coverage metrics).

Note: The `CollectHsMetrics` tool reports the percent of bases covered at certain sequencing depths (e.g., 1X, 10X, 20X, 30X, 40X, and 50X). To obtain coverage for additional sequencing depths ε N use the command "gawk '$1 != "chrom" && $4 >= N' SAMPLE_per_base_coverage_sorted_rmdups.txt | wc -l" to obtain the number of bases with at least N coverage. Divide this number by the "TARGET_TERRITORY" value from SAMPLE_hs_metrics_sorted_rmdups.txt to calculate "% bases ε N". The "SAMPLE_per_base_coverage*.txt" files can be quite large for large designs, and can be compressed once sequencing depths have been calculated using `gzip` as described earlier.

Calculate Coverage in Exonic Target Regions

The percentage of exonic positions with coverage higher than a cutoff is an important metric to evaluate the quality of target enrichment in exonic target regions. It can be calculated by leveraging the "SAMPLE_per_base_coverage_sorted_rmdups.txt" file generated by GATK `CollectHsMetrics`.

Package➔Tool(s) Used	BEDTools➔`intersect`
Input(s)	SAMPLE_per_base_coverage_sorted_rmdups.txt Exon_sorted.bed
Output(s)	SAMPLE_per_base_coverage_sorted_rmdups_exon.bed

Reformat the per Base Coverage File to a BED File

```
gawk -v OFS="\t" 'NR>1 {print $1, $2-1, $2, $4}' SAMPLE_per_base_coverage_sorted_rmdups.txt > SAMPLE_per_base_coverage_sorted_rmdups.bed
```

Select Base Positions in Exonic Regions

```
bedtools intersect -a SAMPLE_per_base_coverage_sorted_rmdups.bed -b /path/to/Exon_sorted.bed -u > SAMPLE_per_base_coverage_sorted_rmdups_exon.bed
```

Additional Levels of Coverage (see note below for calculation)

```
gawk '$4 >= N' SAMPLE_per_base_coverage_sorted_rmdups_exon.bed | wc -l
```

Note: To obtain coverage for additional sequencing depths ε N use the command "gawk '$4 >= N' SAMPLE_per_base_coverage_sorted_rmdups_exon.bed | wc -l" to obtain the number of bases in exonic regions with at least N coverage. Divide this number by the "TARGET_TERRITORY" value from SAMPLE_hs_metrics_sorted_rmdups.txt to calculate "% Panel exon region ≥ N".

Description of Metrics

The tools used in this document generate output files that contain many metrics. There are some metrics that are frequently monitored to assess capture experiment performance. Table 3 and Table 4 describe many of these metrics, which tool(s) are used to generate the metrics, and additional mathematical or string parsing operations that may be necessary to obtain the final values.

Metrics for DNA samples

Metric	Tool(s) used to obtain value (name of output file used)	Description	Metric name in tool's output file and/or calculation method
Total input reads	**fastp (SAMPLE_fastp.log)**	Number of reads prior to fastp processing for quality and adapter trimming	"Read1 before filtering: total reads" + "Read2 before filtering: total read"
Total reads after adapter trimming	**fastp (SAMPLE_fastp.log)**	Number of reads after fastp processing for quality and adapter trimming	"Filtering result: reads passed filter"
% Input reads after filtering	**fastp (SAMPLE_fastp.log)**	Percentage of total input reads remaining after fastp processing	("Filtering result: reads passed filter") / ("Read1 before filtering: total reads" + "Read2 before filtering: total read") * 100
% Reads mapped	**samtools flagstat (SAMPLE_flagstat.txt)**	Percentage of filtered reads that are mapped in the genome	%value in the "mapped" field
% Paired reads mapped	**samtools flagstat (SAMPLE_flagstat.txt)**	Percentage of filtered reads that are paired and mapped in the genome	%value in the "properly paired" field
Barcode/umi-based duplicate rate	**GATK CollectAlignmentSummaryMetrics (SAMPLE_alignment_metrics_sorted.txt)** **GATK CollectAlignmentSummaryMetrics (SAMPLE_alignment_metrics_sorted_rmdups.txt)**	Percentage of aligned reads identified as PCR duplicates (fragments with the same start-end and UMIs)	total_mapped_reads_before_dedup = CollectAlignmentSummaryMetrics PF_READS_ALIGNED in the PAIR column of SAMPLE_alignment_metrics_sorted.txt total_mapped_reads_after_dedup = CollectAlignmentSummaryMetrics PF_READS_ALIGNED in the PAIR column of SAMPLE_alignment_metrics_sorted_rmdups.txt total duplicate rate = (total_mapped_reads_before_dedup - total_mapped_reads_after_dedup) / (total_mapped_reads_before_dedup) * 100

Position-based duplicate rate	**GATK CollectAlignmentSummaryMetrics (SAMPLE_alignment_metrics_sorted.txt)** **GATK CollectAlignmentSummaryMetrics (SAMPLE_alignment_metrics_sorted_rmdups_gatk.txt)**	Percentage of aligned reads identified as positional duplicates (fragments with the same start-end)	total_mapped_reads_before_dedup = CollectAlignmentSummaryMetrics PF_READS_ALIGNED in the PAIR column of SAMPLE_alignment_metrics_sorted.txt total_mapped_reads_after_dedup = CollectAlignmentSummaryMetrics PF_READS_ALIGNED in the PAIR column of SAMPLE_alignment_metrics_sorted_rmdups_gatk.txt total duplicate rate = (total_mapped_reads_before_dedup - total_mapped_reads_after_dedup) / (total_mapped_reads_before_dedup) * 100
Optical duplicate rate	**GATK MarkDuplicates (SAMPLE_markduplicates_metrics_gatk.txt)**	Percentage of aligned reads identified as optical duplicates, includes both paired and unpaired reads	(READ_PAIR_OPTICAL_DUPLICATES * 2) / ((READ_PAIRS_EXAMINED * 2) + UNPAIRED_READS_EXAMINED) * 100
% Reads on-target (before duplicates removal)	**GATK CountReads (ontarget_reads_sorted.txt)** **GATK CollectAlignmentSummaryMetrics (SAMPLE_alignment_metrics_sorted.txt)**	Percentage of mapped reads overlapping a target region by at least 1 base. No padding/buffering.	ontarget_reads = value ("Tool returned:") in ontarget_reads_sorted.txt total_mapped_reads = CollectAlignmentSummaryMetrics PF_READS_ALIGNED in the PAIR column of SAMPLE_alignment_metrics_sorted.txt % mapped reads on-target = (on_target_reads) / (total_mapped_reads) * 100

% Reads on-target (after duplicates removal)	**GATK CountReads (ontarget_reads_sorted_rmdups.txt)** **GATK CollectAlignmentSummaryMetrics (SAMPLE_alignment_metrics_sorted_rmdups.txt)**	Percentage of mapped, non-duplicate reads overlapping a target region by at least 1 base. No padding/buffering.	ontarget_reads = value ("Tool returned:") in ontarget_reads_sorted_rmdups.txt total_mapped_reads = CollectAlignmentSummaryMetrics PF_READS_ALIGNED in the PAIR column of SAMPLE_alignment_metrics_sorted_rmdups.txt % mapped, non-duplicate reads on-target = (on_target_reads) / (total_mapped_reads) * 100
Fold enrichment (after duplicates removal)	**GATK CollectHsMetrics (SAMPLE_hs_metrics_sorted_rmdups.txt)**	Fold enrichment of the capture target compared to the whole genome. In terms of the metrics in the CollectHsMetrics output file: (ON_BAIT_BASES / (ON_BAIT_BASES + NEAR_BAIT_BASES + OFF_BAIT_BASES)) / (BAIT_TERRITORY / GENOME_SIZE). In other words, the fraction of sequencing bases in the capture target divided by the fraction of total genomic bases in the capture target.	FOLD_ENRICHMENT
Median insert size (after duplicates removal)	**GATK CollectInsertSizeMetrics (SAMPLE_insert_size_metrics_sorted_rmdups.txt)**	Median estimated capture fragment insert size	MEDIAN_INSERT_SIZE
Mean insert size (after duplicates removal)	**GATK CollectInsertSizeMetrics (SAMPLE_insert_size_metrics_sorted_rmdups.txt)**	Mean estimated capture fragment insert size	MEAN_INSERT_SIZE
Insert size std dev (after duplicates removal)	**GATK CollectInsertSizeMetrics (SAMPLE_insert_size_metrics_sorted_rmdups.txt)**	Standard deviation of the estimated capture fragment insert size	STANDARD_DEVIATION

Mean target coverage (before duplicates removal)	**GATK CollectHsMetrics (SAMPLE_hs_metrics_sorted.txt)**	Mean depth of coverage over the capture target	MEAN_TARGET_COVERAGE
Mean target coverage (after duplicates removal)	**GATK CollectHsMetrics (SAMPLE_hs_metrics_sorted_rmdups.txt)**	Mean depth of coverage over the capture target	MEAN_TARGET_COVERAGE
Median target coverage (before duplicates removal)	**GATK CollectHsMetrics (SAMPLE_hs_metrics_sorted.txt)**	Median depth of coverage over the capture target	MEDIAN_TARGET_COVERAGE
Median target coverage (after duplicates removal)	**GATK CollectHsMetrics (SAMPLE_hs_metrics_sorted_rmdups.txt)**	Median depth of coverage over the capture target	MEDIAN_TARGET_COVERAGE
% Bases in N-fold range (e.g., N=2 or 10)	**GATK CollectHsMetrics (SAMPLE_hs_metrics_sorted_rmdups.txt)** **"gawk '$1 != "chrom" && $4 >= MEDIAN_TARGET_COVERAGE/N && $4 <= MEDIAN_TARGET_COVERAGE*N' SAMPLE_per_base_coverage_sorted_rmdups.txt | wc -l"/TARGET_TERRITORY * 100**	Percentage of capture target bases covered by between (MEDIAN_TARGET_COVERAGE / N) and (MEDIAN_TARGET_COVERAGE * N) reads after duplicates removal	extract values of MEDIAN_TARGET_COVERAGE and TARGET_TERRITORY from SAMPLE_hs_metrics_sorted_rmdups.txt calculate “% Bases in N-fold range” using formula on the left
% Bases > 0.2-fold of unique depth	**GATK CollectHsMetrics (SAMPLE_hs_metrics_sorted_rmdups.txt)** **"gawk '$1 != "chrom" && $4 >= 0.2*MEDIAN_TARGET_COVERAGE' SAMPLE_per_base_coverage_sorted_rmdups.txt | wc -l"/TARGET_TERRITORY * 100**	Percentage of capture target bases covered by more than (0.2 * MEDIAN_TARGET_COVERAGE) reads after duplicates removal	extract values of MEDIAN_TARGET_COVERAGE and TARGET_TERRITORY from SAMPLE_hs_metrics_sorted_rmdups.txt calculate “% Bases > 0.2-fold of unique depth” using formula on the left
% Panel exon region ≥ N (e.g., N=300 or 1000)	**GATK CollectHsMetrics (SAMPLE_hs_metrics_sorted_rmdups.txt)** **"gawk '$4 >= N' SAMPLE_per_base_coverage_sorted_rmdups_exon.bed | wc -l"/TARGET_TERRITORY * 100**	Percentage of exonic capture target bases covered by more than N reads after duplicates removal	extract values of TARGET_TERRITORY from SAMPLE_hs_metrics_sorted_rmdups.txt calculate “% Panel exon region ≥ N” using formula on the left
Error rate	**samtools stats (SAMPLE_stats.txt)**	Ratio between mismatches and bases mapped	extract value of “error rate” from SAMPLE_stats.txt

GE recovery rate	**GATK CollectHsMetrics (SAMPLE_hs_metrics_sorted_rmdups.txt)**	Genome equivalents recovered in sequenced library / input genome equivalents	extract value of MEDIAN_TARGET_COVERAGE from SAMPLE_hs_metrics_sorted_rmdups.txt calculate “GE recovery rate” using MEDIAN_TARGET_COVERAGE / (input mass * 330)

Table 3. Description of important metrics for DNA samples

Metrics for RNA samples *(not applicable for the AVENIO Edge System)*

Metric	Tool(s) used to obtain value (name of output file used)	Description	Metric name in tool's output file and/or calculation method
Total input reads	**fastp (SAMPLE_fastp.log)**	Number of reads prior to fastp processing for quality and adapter trimming	"Read1 before filtering: total reads" + "Read2 before filtering: total read"
Total reads after adapter trimming	**fastp (SAMPLE_fastp.log)**	Number of reads after fastp processing for quality and adapter trimming	"Filtering result: reads passed filter"
% Input reads after filtering	**fastp (SAMPLE_fastp.log)**	Percentage of total input reads remaining after fastp processing	("Filtering result: reads passed filter") / ("Read1 before filtering: total reads" + "Read2 before filtering: total read")*100
Number of reads after rRNA removal	**bwa, samtools (SAMPLE_rRNA_read_counts.tsv)**	Number of reads after removing those from rRNA	Number of reads after rRNA removal
% Uniquely mapped reads after rRNA removal	**STAR alignReads (SAMPLE_star_Log.final.out)**	Percentage of reads that are uniquely mapped after rRNA removal	Uniquely mapped reads %
% Reads on-target for entire panel	**bedtools, samtools (SAMPLE_on_target_rate.tsv)**	Percentage of mapped reads overlapping a target region by at least 1 base	% Reads on-target for entire panel
% Reads on-target housekeeping genes	**bedtools, samtools (SAMPLE_on_target_rate.tsv)**	Percentage of mapped reads overlapping a target region of housekeeping genes by at least 1 base	% Reads on-target housekeeping genes
% Reads on-target fusion genes in design	**bedtools, samtools (SAMPLE_on_target_rate.tsv)**	Percentage of mapped reads overlapping a target region of fusion genes in design by at least 1 base	% Reads on-target fusion genes in design
% rRNA reads	**bwa, samtools (SAMPLE_rRNA_read_counts.tsv)**	Percent of reads that are mapped to rRNA reference	% rRNA reads

Table 4. Description of important metrics for RNA samples

3. Reference Links

Roche is not responsible for the content of the following third-party websites.

- BWA: *https://github.com/lh3/bwa*
- FastQC: *http://www.bioinformatics.babraham.ac.uk/projects/fastqc/*
- GATK (Broad Institute): *https://software.broadinstitute.org/gatk/*
- SAMtools: *http://www.htslib.org/*
- BEDTools: *https://github.com/arq5x/bedtools2*
- seqtk: *https://github.com/lh3/seqtk*
- fastp: *https://github.com/OpenGene/fastp*
- fgbio: *https://github.com/fulcrumgenomics/fgbio*
- VarDict: *https://github.com/AstraZeneca-NGS/VarDictJava*
- VarScan2: *http://varscan.sourceforge.net/index.html*
- MSIsensor: *https://github.com/ding-lab/msisensor/blob/master/README_msisensor.md*
- MSIsensor2: *https://github.com/niu-lab/msisensor2*
- CNVkit: *https://github.com/etal/cnvkit*
- STAR: *https://github.com/alexdobin/STAR*
- STAR-Fusion: *https://github.com/STAR-Fusion/STAR-Fusion/wiki*
- CTAT-Splicing: *https://github.com/NCIP/CTAT-SPLICING/releases*

4. Glossary

BAI file – BAM index file. For tools that require an indexed **BAM file,** the BAI file must be present in the same location as the BAM file.

Bait interval (GATK) – See **Capture target**.

BAM file – Compressed form of the **SAM file** format.

BED file – File format for describing genomic regions/intervals. BED file start coordinates are 0-based.

bp – Abbreviation for base pair.

Capture target – as defined by Roche, these are the regions covered directly by one or more probes or primer pairs. These are equivalent to the **Bait intervals** referred to by GATK.

FASTA file – A standard file format for describing nucleic acid sequences.

FASTQ file – A standard file format for describing sequencing reads that also includes base quality information.

Genomic index – A form of the reference genome sequence that enables faster comparisons during alignment.

Interval file – File format for describing genomic regions/intervals that also contains a header describing the reference genome. Genomic interval file start coordinates are 1-based. See **Bait interval (GATK)** and **Target interval (GATK)**.

Primary target – as defined by Roche, these are the regions against which probes or primer pairs were designed. Regions with no probes or primer pairs targeting them are excluded. These are equivalent to the **Target intervals** referred to by GATK.

SAM file – Sequence Alignment / Map file; a community standard format for specifying sequencing read alignment to a reference genome.

Target interval (GATK) – see **Primary target**.

Target region – see **Primary target**.

VCF file – Variant call format; a community standard format for specifying variant calls for one or more samples or populations against a reference genome.

5. REFERENCES

- Jia, Peng. "MSIsensor-pro: Fast, Accurate, and Matched-normal-sample-free Detection of Microsatellite Instability." *Genomics, Proteomics & Bioinformatics*, vol. 18, no. 1, 2020, pp. 65-71.
- Niu, Beifang. "MSIsensor: microsatellite instability detection using paired tumor-normal sequence data." *Bioinformatics*, vol. 30, no. 7, 2014, pp. 1015-1016.
- Talevich, Eric. "CNVkit: Genome-Wide Copy Number Detection and Visualization from Targeted DNA Sequencing." *Plos Computational Biology*, 2016.

Published by:
Roche Sequencing Solutions, Inc.
4300 Hacienda Drive
Pleasanton, CA 94588

MC—08068 11/22