Supplementary Material for:

DeepClone, an end-to-end protocol to study somatic mutagenesis and selection at high resolution

Authors

Ferriol Calvet[1,2,3], Morena Pinheiro-Santin[1], Erika López-Arribillaga[1,3], Raquel Blanco Martinez-Illescas[1,2,3], Núria Samper[1], Miguel L. Grau[1], Ferran Muiños[1,3], Rocío Chamorro González[1], Maria Andrianova[1], Federica Brando[1], Stefano Pellegrini[1,2,3], Axel Rosendahl Huber[1,3], Marta Huertas[1], Elisabet Figuerola-Bou[1], Coohleen Coombes[4], Brendan F. Kohrn[4], Jeanne Fredrickson[4], Rosa Ana Risques[4], Nuria Lopez-Bigas[1,2,3,5,6,@], Abel Gonzalez-Perez[1,2,3,6,@]

[6] These authors jointly supervised this work: A. Gonzalez-Perez, N. Lopez-Bigas

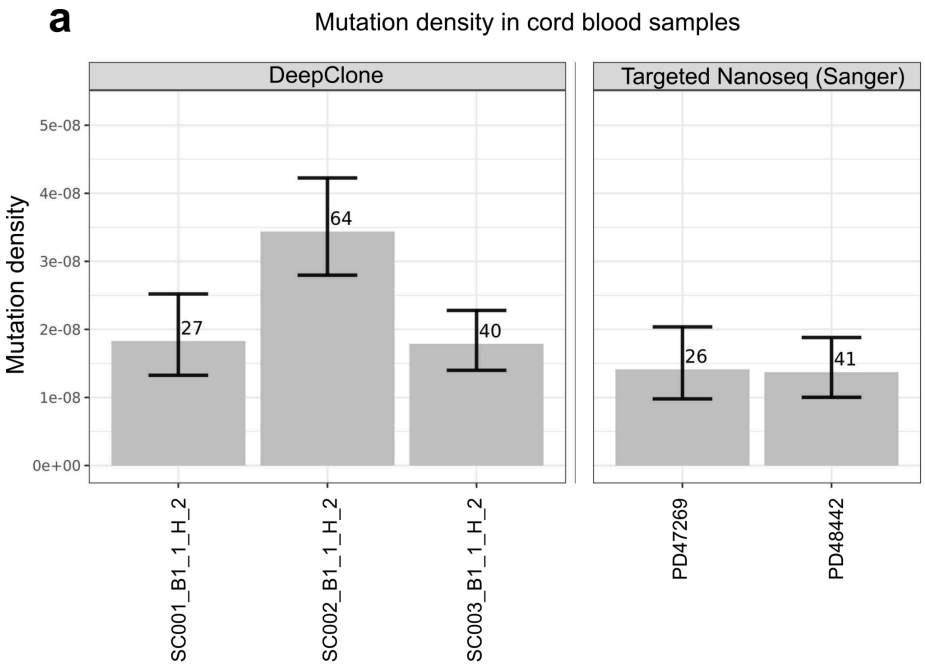
[@] Correspondence should be addressed to Nuria Lopez-Bigas <nuria.lopez@irbbarcelona.org> and Abel Gonzalez-Perez <abel.gonzalez@irbbarcelona.org>

Affiliations

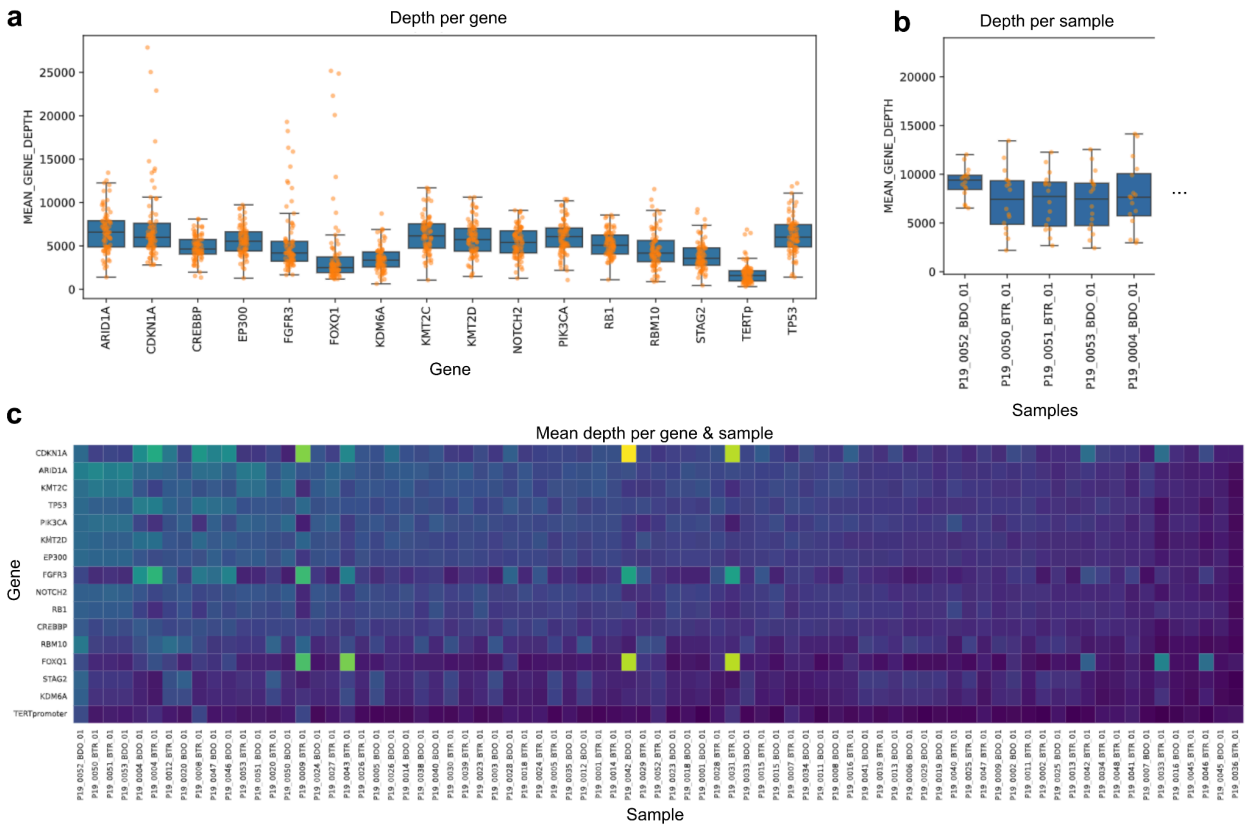
1. Institute for Research in Biomedicine (IRB Barcelona), The Barcelona Institute of Science and Technology, Baldiri Reixac, 10, 08028 Barcelona, Spain.
2. Department of Medicine and Life Sciences, Universitat Pompeu Fabra, Barcelona, Spain.
3. Centro de Investigación Biomédica en Red en Cáncer (CIBERONC), Instituto de Salud Carlos III, Madrid, Spain.
4. Department of Laboratory Medicine and Pathology, University of Washington, Seattle, WA.
5. Institució Catalana de Recerca i Estudis Avançats (ICREA), Barcelona, Spain.

Table of Contents

Extended Data Figures



Extended Data Figure 1. Error rate of the DNA duplex library preparation protocol.
The bars represent the mutation density (mutations per Mbps) calculated from independent cord blood samples. The difference between this number and that expected in the cord blood, of around $2x10^{-8}$ mutations per Mb (obtained from experiments using expanded hematopoietic stem cells[1,2]) represents the estimation of the error rate of the technology.
The error rate of the DNA duplex library preparation protocol in DeepClone, left (using the IDT cfDNA kit) and that of the targeted Nanoseq protocol[3], right are represented.

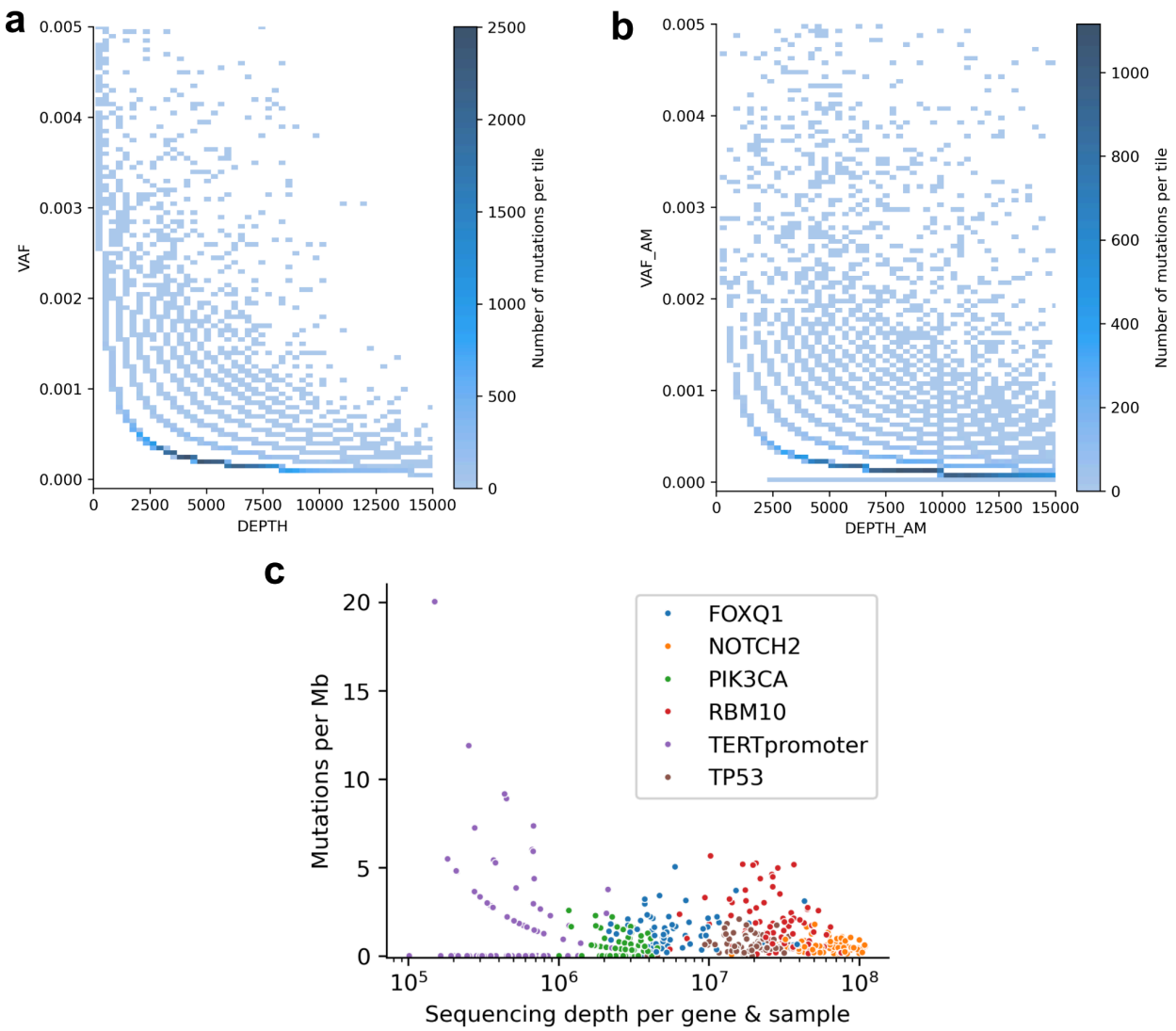


Extended Data Figure 2. Exploration of DNA duplex sequencing depth across samples in a cohort using deepCSA.

a. Average duplex sequencing depth for every gene (and the TERT promoter) across normal urothelium samples (dots in the boxplot).

b. Average duplex sequencing depth obtained in several normal urothelium samples (each dot represents a gene).

c. Average duplex sequencing depth per gene and sample in the same cohort.

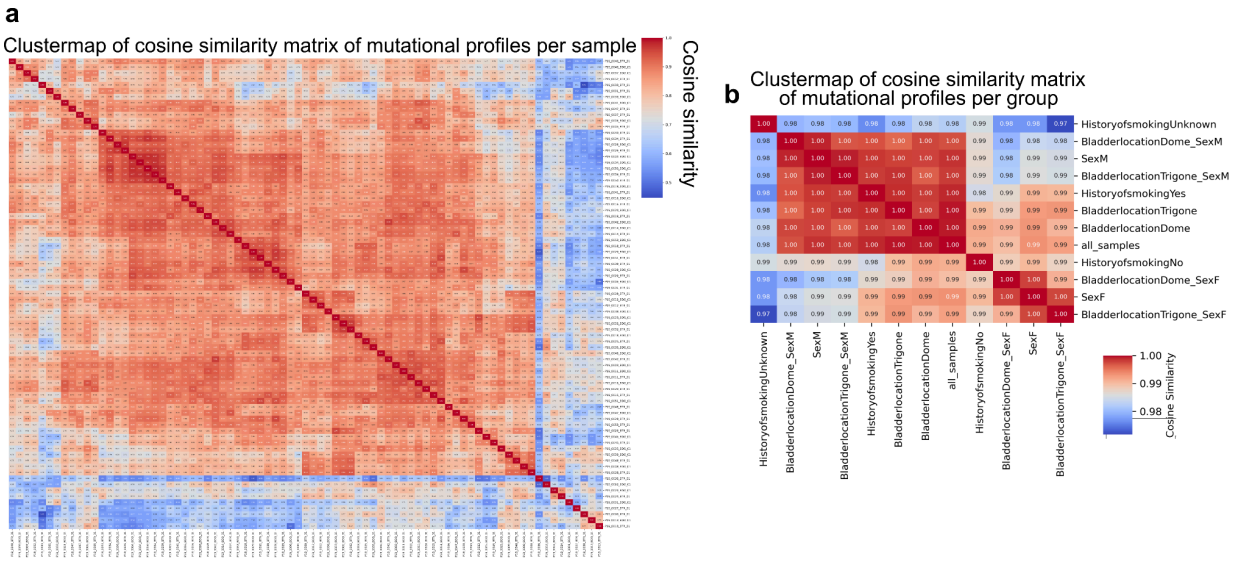


Extended Data Figure 3. Effect of different sequencing depths on the VAF and the mutation density.

a. Variant allele frequency of mutations detected with different duplex sequencing depths. The counts of reads supporting the reference and alternate nucleotides, in this case, are obtained using only information from duplex quality reads. Lower depths can cause the variant allele frequency of the mutation to be overestimated. This is why deepCSA provides the All molecules VAF, which uses not only reads that form duplex, but also those that are left at the status of single strand consensus (b).

Tiles represent a region in the VAF vs depth space, and have the same size (300 reads of depth x 0.00005 VAF) in both plots.

c. Relationship between the mutation density of 5 genes and the TERT promoter sequenced across normal urothelium samples and the average sequencing depth of different genes. Every dot in the plot represents a gene in a sample. For the smaller genomic elements (such as the TERT promoter or FOXQ1), the mutation density is overestimated for samples with lower depth.

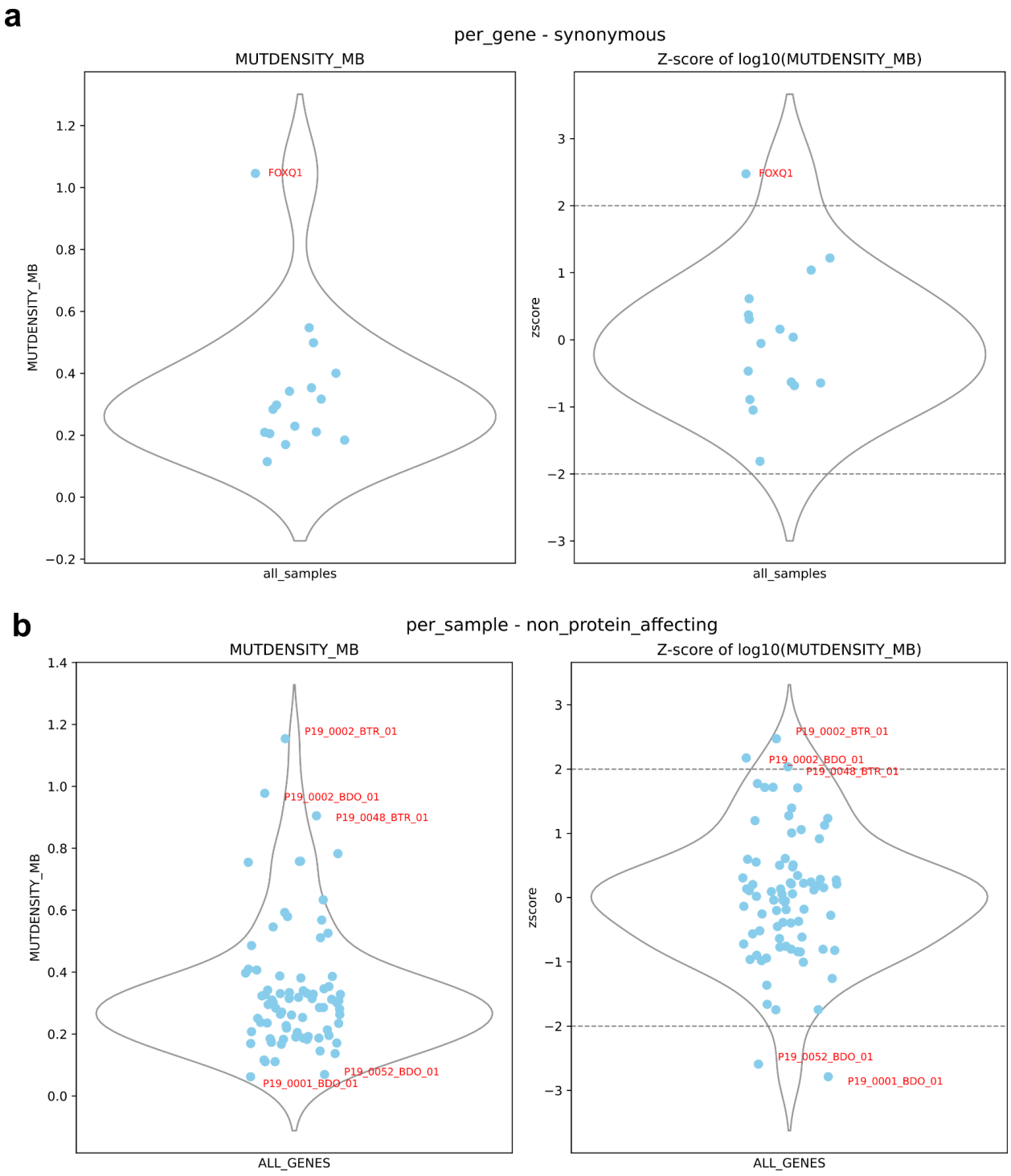


Extended Data Figure 4. Similarity of the mutational profile of different urothelium samples.

a. The heatmap represents the cosine similarity (in a blue-to-red color scale) of the mutational profile of pairs of urothelium samples. While the vast majority of samples show a highly coherent profile, implying a set of underlying mutational processes, a few show a divergent profile which should be analyzed further to understand whether it is the product of highly prevalent artifacts or a different history of exposures.

b. In this heatmap, the cosine similarity between pre-defined groups of samples (e.g., by history of exposure, sex, or other features).

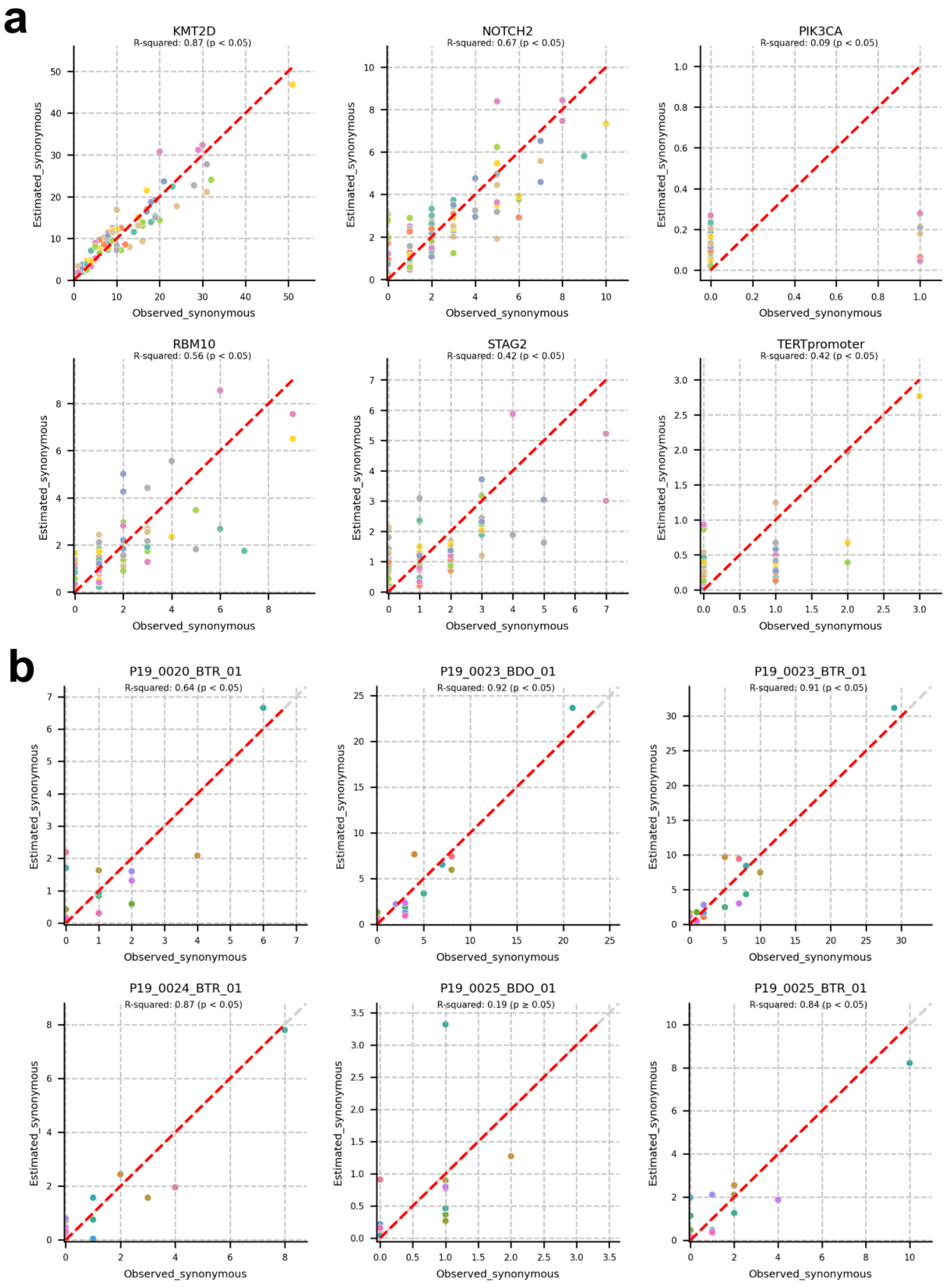
Both plots are automatically generated in the concat profiles step of deepCSA.



Extended Data Figure 5. Quality control of mutation density values across genes and samples.

a,b. Distribution of the synonymous mutation density (a) and the Z-score (b) of all genes sequenced across normal urothelium samples (each dot, a gene). The Z-score is calculated using the mean mutation density of each gene and the distribution of mutation density observed for all genes. The synonymous mutation density of FOXQ1 is more than two standard deviations

above the mean of the distribution of all genes. This points at potential problems, such as systematically lower sequencing depth (see Extended Data Fig. 3).
c,d. Distribution of the non-protein affecting mutation density (a) and the Z-score (b) of all normal urothelium samples in a cohort (each dot, a sample). The Z-score is calculated using the mean mutation density of each sample and the distribution of mutation density observed for all samples. Samples with Z-score more than two standard deviations above (or more than two standard deviations below) the mean of the distribution of all samples should be checked (e.g., the higher mutation rate of the first group could be due to different exposures), before proceeding to downstream analyses.

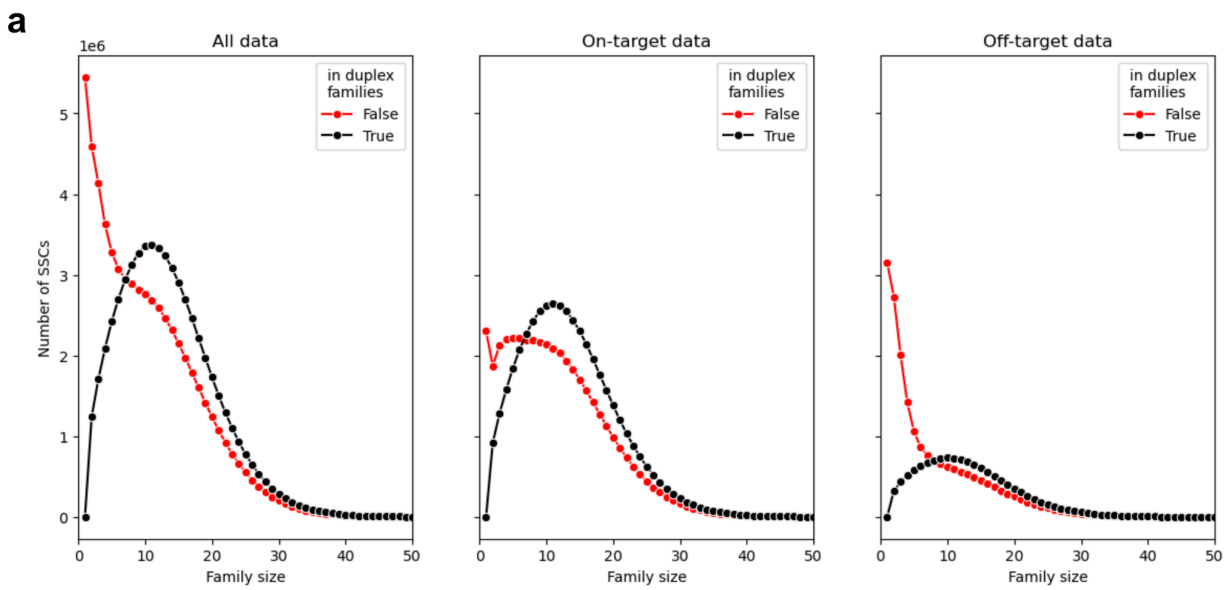

a
KMT2D
R-squared: 0.87 (p < 0.05)
NOTCH2
R-squared: 0.67 (p < 0.05)
PIK3CA
R-squared: 0.09 (p < 0.05)
RBM10
R-squared: 0.56 (p < 0.05)
STAG2
R-squared: 0.42 (p < 0.05)
TERTpromoter
R-squared: 0.42 (p < 0.05)
b
P19_0020_BTR_01
R-squared: 0.64 (p < 0.05)
P19_0023_BDO_01
R-squared: 0.92 (p < 0.05)
P19_0023_BTR_01
R-squared: 0.91 (p < 0.05)
P19_0024_BTR_01
R-squared: 0.87 (p < 0.05)
P19_0025_BDO_01
R-squared: 0.19 (p ≥ 0.05)
P19_0025_BTR_01
R-squared: 0.84 (p < 0.05)
Estimated_synonymous
Observed_synonymous

Extended Data Figure 6. Omega values calculated using smoothed synonymous mutations density.

DeepCSA implements a calculation of omega that is based on smoothed synonymous mutations (across all samples) rather than using the synonymous mutations actually observed in the sample in question. This smoothing of the synonymous mutation density allows the calculation of omega values even for genes with a low number of observed synonymous mutations in a given sample. These smoothed omega values need to be checked versus their expectation using the observed synonymous mutations, to guarantee that no biases are introduced in their calculation.

a. These plots show the agreement between the values of omega calculated for five genes and the TERT promoter (non activating mutations in this case) in each normal urothelium sample using the smoothed synonymous mutation counts (y-axis) and the value of observed number of synonymous mutations (x-axis), in all the cases that this can be calculated. The longer the sequence of the gene included in the hybridization capture, the higher the agreement between the smoothed omega value and the value calculated from the observed synonymous mutations. In the cases where the agreement is poor, the smoothed omega value cannot be used to estimate the strength of positive selection on the protein affecting mutations in the gene in the sample.

b. Same as a, for six exemplary normal urothelium samples, that is, pooling all synonymous mutations observed in the sample across genes. The same trends are observed at the sample level.



Extended Data Figure 7. Distribution of family size of unique molecules across all, targeted, and off-target regions. From left to right, all unique molecules, only unique molecules mapping to the targeted regions, only unique molecules not mapping to the targeted regions. The plots follow the same rationale as those in Figure 4 of the main manuscript, but representing the number of unique molecules instead of the proportion of reads. Off-target unique molecules tend to be less amplified and also are less likely to form duplex families. This is expected since the capture and amplification of the two independent strands of a given genomic fragment are less likely to reach the final sequenced library because they should not be captured by the hybridization panel.

Supplementary Tables

Supplementary Table 1. Comparison of DNA duplex sequencing and analysis protocols

Method	**Duplex Seq**	**Targeted Nanoseq**	**UDSeq**	**DeepClone**
Reference	Refs[4–6]	Refs[3,7]	Ref[8]	Ref[9]
DNA duplex library preparation	Sonication/Enzymatic fragmentation; NEB reagents; custom duplex adapters annealed in-house	UltraShear fragmentation; NEB reagents; xGen CS Adapter - Tech Access (IDT); addition of ddBTPs and no End-Repair	Fragmentase/UltraShear fragmentation; IDT cfDNA kit (on beads)	UltraShear fragmentation; IDT cfDNA kit; intra-strand repair using nucleotide glycosylases
Calling	In house scripts and external code[5]	Targeted Nanoseq pipeline https://github.com/cancerit/NanoSeq	DupCaller pipeline and external tools https://github.com/AlexandrovLab/DupCaller	deepUMIcaller https://github.com/bbglab/deepUMIcaller
Analysis of mutagenesis and selection	In house scripts https://github.com/risqueslab/DuplexSequencingScripts	dNdScv (for dN/dS estimates)	DupCaller (for mutation burden)	deepCSA https://github.com/bbglab/deepCSA
Max depth reported per sample	In theory, more than 20,000x[5]	~660x on average[3]	No examples provided	~3,500x on average[9]
Error rate	~4×10^{-8} [5]	5×10^{-9} [3]	2.5×10^{-9} [8]	Extended Data Fig. 1 ~2.5×10^{-8}
Applied in studies	Refs[5,6,10–12]	Refs[3,13]	Refs[8,14]	Experimental protocol[15] Computational pipeline[9]

Supplementary Table 2. Selected metrics calculated within DeepClone

Metric	Type of metric	Description
WetLab>>Input (ng)	Recovery	Initial DNA amount used for library prep in ng. Recoveries are calculated based on this initial input value.
WetLab>>DNA after ligation (ng)	Recovery	DNA amount recovered after ligation in ng. Used to calculate mass recovery at the ligation step.
WetLab>>fmol to PCR1	Recovery	Number of unique molecules in fmol measured by qPCR used as an input for the Indexing PCR. It is an estimation of the maximum diversity of the duplex library preparation.
WetLab>>Recovery Input to ligation	Recovery	Mass recovery of DNA from input to ligation. Considers total mass recovery including both ligated and non ligated DNA.
WetLab>>Recovery Input to ligation qPCR	Recovery	Recovery of ligated DNA measured by qPCR.
DryLab>>On-target unique molecules percentage	Duplex library prep. performance	Proportion of unique molecules that map to the capture panel regions. This value can vary depending on the number of capture steps and the size of the panel, but generally it should be greater than 0.7.
DryLab>>Raw to DSC	Duplex library prep. performance	Total number of unique molecules divided by the number of double strand consensus (DSC) families. Approximation of how many raw reads on average are required to form a duplex quality DSC.
DryLab>>SSC to DSC	Duplex library prep. performance	Number of single strand consensus (SSC) families divided by the number of double strand consensus (DSC) families.
DryLab>>Family size	Sequencing estimation	Peak size of the distribution of family sizes (number of sequencing reads of the same unique molecule that form a family).
DryLab>>GBs analysed	Sequencing estimation	GBs of data analysed.
DryLab>>Total GBs for optimal	Sequencing estimation	Estimated GBs required to reach optimal sequencing defined by an average of 18x amplification for each on-target molecule.
DryLab>>GBs (analyzed-optimal)	Sequencing estimation	Missing GBs to reach optimal sequencing. It is a subtraction of the GBs analysed from the Total GBs for optimal.
DryLab>>Percentage of duplicates on-target	Summary stats	Percentage of reads that are copies of on-target unique molecules.
DryLab>>Amplificati	Sequencing	Ratio of on-target molecules amplified versus off-target molecules

on bias on/off	estimation	amplified. Indicates selectivity: a higher ratio means on-target molecules are more amplified compared to off-target.
DryLab>>Depth	Summary stats	Duplex sequencing depth in the regions provided as targets. Be aware that the 250bp added on each target flanking ends may lead to this value being underestimated.
Combined>>Unique molecules sequenced vs qPCR	Sequencing estimation	Ratio of unique DNA fragments sequenced versus estimated by qPCR. The closer this value is to 1, the better the qPCR estimation of the unique DNA fragments was.
Combined>>Recovery input to depth	Recovery	Recovered duplex depth normalized by the input DNA quantity, which provides an estimation of the overall yield and efficiency of the protocol.

Supplementary Note

Implementation of an alternative DNA duplex library preparation protocol

We have tested a duplex library preparation protocol that is different from that described in the manuscript. The main difference consists in the type of duplex adapters used (xGen CS Adapter - Tech Access, IDT, Ref. 1080799). These duplex adapters have a T overhang in the 3' end of their sequence and therefore, this protocol would require an A-tailing step. These modifications in the procedure are detailed below. The main advantage of this protocol to the one described in the manuscript is the ligation chemistry, which requires a single ligation step instead of the ligation followed by sequencing, thus potentially increasing the overall efficiency of the library preparation. Furthermore, there are no limitations to the maximum DNA input as in the xGen™ cfDNA & FFPE DNA Library Prep kit. Nevertheless, unlike the library preparation protocol described in the manuscript, using these duplex adapters precludes the use of a ready-made kit. Instead, reagents for each step need to be acquired separately to implement the protocol, which is described below.

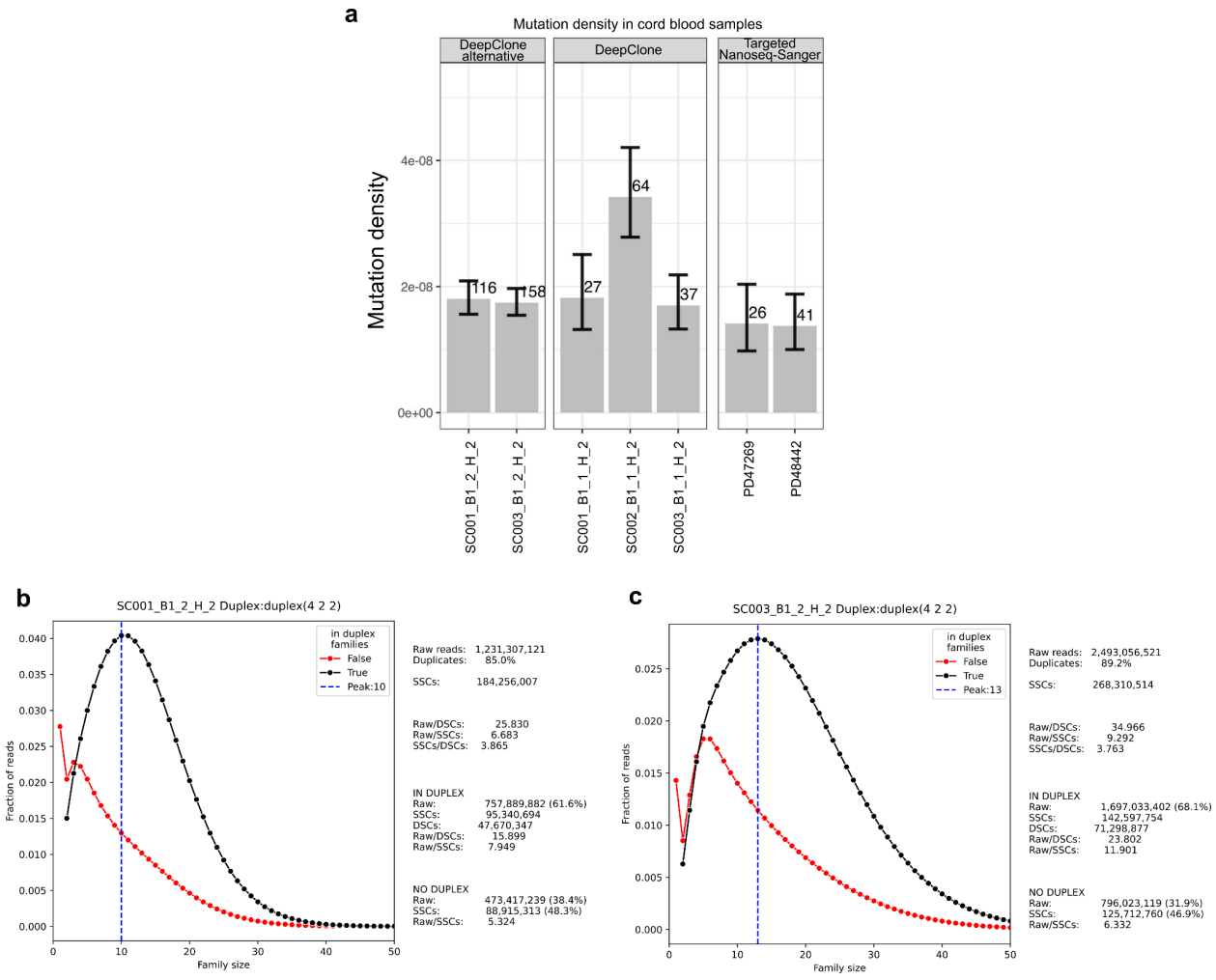
Procedure

Starting DNA is diluted in a final volume of 26 µL EB buffer. Fragmentation is performed by adding 14 µL NEBuffer r1.1 (NEB, Ref. B7030S), 4 µL NEBNext UltraShear Enzyme (NEB, Ref. M7634S) and 1.4 µL 50 mM NAD+ (NEB, Ref. B9007S). The reaction is incubated at 37ºC for an initial 20 minutes, informed by previous kinetics with genomic DNAs at similar integrity status. The same criteria for optimal fragmentation is applied as in the DeepClone duplex library preparation protocol. After optimal fragmentation is achieved, the reaction is inactivated in a pre-heated thermocycler at 65°C for 15 minutes (hold at 4ºC), with the lid temperature set to 75ºC. DNA is purified by performing a 1.8× AmpureXP bead clean-up (Beckman Coulter, Ref. A63882). DNA is eluted in 52 µL of EB buffer and 50 µL of the cleaned DNA is taken into the End-Repair and A-tailing reaction, consisting of 7 µL of NEBNext Ultra II End Prep Reaction Buffer, 3 µL of NEBNext Ultra II End Prep Enzyme Mix (NEB, Ref. E7546S). End-Repair and A-tailing is performed by incubating at 37ºC for 30 minutes, followed by 65ºC for 30 minutes (hold at 4°C) with the thermocycler lid set to 75°C. Ligation mix consisting of 30 µL NEBNext Ultra II Ligation Master Mix (NEB, Ref. E7595S), 1 µL NEBNext Ligation Enhancer (NEB, Ref. E7595S), 11 µL nuclease-free water and 2.5 µL xGen CS Adapter (IDT, Ref. 1080799) is added to the End-Repair and A-tailing reaction. Ligation was performed by incubating at 20ºC for 15 minutes with the thermocycler lid temperature tuned off. DNA is purified with a 0.8x AmpureXP bead ratio (Beckman Coulter, Ref. A63882), and eluted in 25 µL of EB buffer. The whole reaction is taken into Intra-strand repair, as described in the DeepClone duplex library preparation protocol. All remaining steps are carried out as in the main procedure.

Preliminary results

The application of this alternative library preparation experimental protocol to two cord blood DNA samples (as described in “Rate of errors of the technology” section below), yielded DNA duplex libraries that sequenced achieving a performance comparable to that of DNA duplex

sequencing methods described in the main manuscript. In detail, the mutation density obtained from sequencing cord blood samples was similar to that obtained using the experimental duplex library preparation protocol described in the DeepClone main manuscript, and also similar to that reported by targeted Nanoseq suggesting that the error rate of the technology is comparable to these other library preparation approaches (Supp. Data Fig. 1a). The conversion from the starting amount of DNA to the final duplex depth is also comparable to other approaches, as is the sequencing efficiency –the required amount of raw reads to obtain a duplex read (Supp. Data Fig. 1b,c). These preliminary results suggest that this alternative library preparation approach allows the reduction in sequencing cost for the same duplex depth. Another salient advantage of this alternative is that the starting amount of DNA is not limited to 250 ng (as it was the case for the main DeepClone protocol), allowing more genomic DNA input into the duplex library (being 500 ng the maximum we have tried currently), and therefore higher duplex depth per library preparation (see "Experimental Design" section in the main manuscript).



Supplementary Data Figure 1. Overview of the alternative duplex protocol results.
a. Error rate of the DNA duplex library preparation protocol. The bars represent the mutation density (mutations per Mbps) calculated from independent cord blood samples. The difference

between this number and that expected in the cord blood, of around $2x10^{-8}$ mutations per Mb (obtained from experiments using expanded hematopoietic stem cells[1,2]) represents the estimation of the error rate of the technology. The error rate of the DNA duplex library preparation protocol in the alternative DeepClone protocol (explained above using several commercial reagents) is represented on the left, the results from the main DeepClone protocol (using the IDT cfDNA kit) are represented in the middle box and those of the targeted Nanoseq protocol[3] are in the right-most box.
b-c Distribution of family sizes of unique molecules in 2 DNA duplex libraries prepared from different samples. Unique molecules are separated depending on whether (red) or not (black) they form duplex consensus reads. A vertical blue line represents the mode of the SSCs family sizes. The value of this mode (family size peak) is presented within the legend box. Numbers at the right side of the plots present key metrics of the duplex consensus building process. These plots are generated by the deepUMIcaller family metrics module (see FAMILYMETRICS deepUMIcaller step 11b,d in the Supp. Note). The version of these plots for the main DeepClone protocol can be found in Figure 3 in the main text.

Extended protocol for deepUMIcaller

This protocol describes the step-by-step execution of the deepUMIcaller pipeline (github.com/bbglab/deepumicaller).
DeepUMIcaller is an end-to-end computational pipeline that receives DNA duplex sequencing FASTQ files and outputs VCF files with mutations identified across sequenced samples. It is based on an early version of nf-core/fastquorum pipeline, implementing the `fgbio` Best Practices FASTQ to Consensus Pipeline (https://github.com/fulcrumgenomics/fgbio/blob/main/docs/best-practice-consensus-pipeline.md), and a variant calling based on `VarDictJava` (https://github.com/AstraZeneca-NGS/VarDictJava). A series of filters to discard potential artifacts are included in the pipeline. In the following paragraphs, we describe the use of this pipeline to call mutations on the ultradeep sequenced urothelium samples included in this project. All parameters described correspond to this work, but are configurable by the users of the pipeline.

Requirements

Computational:

- High-performance computing cluster
- Maximum resources per task:
    - CPUs: 56 cores
    - Memory: 256 GB
    - Time: 240 hours
- Total amount of storage used by temporary directories: approximately 5 times the initial FASTQ size (in GBs)
- Container Technology: Singularity/Apptainer enabled

Input data:

- input.csv file
    - One row per pair of FASTQs, with sample name, fastq_1, fastq_2 and read_structure.
- Reference genome:
    - GRCh38 reference genome (https://lh3.github.io/2017/11/13/which-human-reference-genome-to-use), mask known artifactual regions of this genome assembly using BEDtools to avoid artifactual alignments or mutations.
    - GRCm39
    - Or any other genome of the sequenced species.
- BED file of the targeted regions: take your targeted file and expand 250bp on each boundary, this way the on-target metrics will be more accurate.

Procedure

Phase 1: Read preprocessing and QCs

1. Samplesheet Validation (INPUT_CHECK:SAMPLESHEET_CHECK). Validates samplesheet format considering the defined deepUMIcaller starting step

```
check_samplesheet.py NOVOGENE_61_to_70_colon_new_ids_SET2.csv \
    samplesheet.valid.csv \
    --step mapping
```

2. Target Region Preparation (BEDTOINTERVAL). Convert BED file to Picard IntervalList format for downstream processing.

```
picard -Xmx3276M BedToIntervalList \
    --INPUT <panel>.bed \
    --OUTPUT <panel>.interval_list \
    --SEQUENCE_DICTIONARY <reference>.dict \
    --TMP_DIR .
```

3. FastQC Quality Control (FASTQC). Generate quality metrics for raw FASTQ files before processing.

```
fastqc --quiet --threads 16 \
    <sample>_1.fastq.gz \
    <sample>_2.fastq.gz
```

4. FASTQ to Unmapped BAM with UMI Extraction (FASTQTOBAM). Process the raw reads to extract the UMI tags using fgbio tools. These will now be stored as BAM tags. The duplex UMI structure has to be defined in the read_structure column of the input CSV file. The output of this step is an unmapped BAM file with UMIs in the RX tag.

```
fgbio -Xmx16g \
    --tmp-dir=. \
    --async-io=true \
    --compression=1 \
    FastqToBam \
    --input <R1>.fq.gz <R2>.fq.gz \
    --output <sample>.unmapped.bam \
    --read-structures 8M+T 8M+T \
```

```
    --sample <sample_id> \
    --library <library_id>
```

Explanation of the 8M+T read structure associated with the duplex adapters used in the experimental part of DeepClone:
8M+T: 8 bases are molecular barcode (M), followed by template sequence (T)
Applied to both R1 and R2
UMIs stored in RX tag of BAM record
<sample_id> and <library_id> are both the name of the sample defined in the sample_name column of the deepUMIcaller input.csv file.

5. Read Trimming (TRIMBAM): This step takes the unmapped BAM and removes the first N bases to trim low-quality bases from read ends. The number of bps to be trimmed from each end of the read can be defined via parameters left_clip and right_clip in the nextflow.config.

```
bam trimBam \
    <sample>.unmapped.bam \
    <sample>.bam \
    -L 10 \
    -R 0
```

6. Alignment to Reference Genome (ALIGNRAWBAM): This step aligns trimmed reads to reference genome while maintaining UMI information tags (RX tag in the BAM file). This maintenance of the tags is ensured by ZipperBAMs fgbio tool.

```
FASTA=`find -L ./ -name "*.amb" | sed 's/.amb//'`

samtools fastq <sample>.bam | \
bwa mem -5 -L 1,1 -K 100000000 -t 32 -p -Y $FASTA - | \
fgbio -Xmx4g \
    --compression 1 \
    --async-io=true \
    ZipperBams \
    --input /dev/stdin \
    --unmapped <sample>.bam \
    --ref $FASTA \
    --output <sample>.mapped.bam \
    --tags-to-reverse Consensus \
    --tags-to-revcomp Consensus
```

7. BAM Sorting and Indexing (SORTBAM). Sort aligned BAM by genomic coordinates and create indexes for rapid access.

```
samtools sort --write-index \
    -@ 16 \
    -o <sample>.sorted.bam \
    -T tmp/ \
    <sample>.mapped.bam
```

8. Quality Control of Aligned Data using QualiMap BAM QC (QUALIMAPQCRAW). This step generates quality metrics for the BAM files with the aligned raw reads. See output section below for more details on the output files.

```
qualimap --java-mem-size=163840M \
```

```
    bamqc \
    --paint-chromosome-limits \
    --genome-gc-distr HUMAN \
    --skip-dup-mode 0 \
    -outformat HTML \
    -bam <sample>.sorted.bam \
    --gff <target_regions>.bed \
    -p non-strand-specific \
    --collect-overlap-pairs \
    -outdir <sample>.raw \
    -nt 2
```

9. Sort reads by template coordinate (SORTBAMCLEAN). This step sorts the reads based on its template coordinate as it is required by the GroupReadsByUmi tool that goes next in the process.

```
samtools sort   --template-coordinate --write-index -@ 16 -o <sample>.resorted.bam -T tmp/<sample>.sorted.bam
```

Phase 2: Duplex consensus building process & all molecules filtering

10. Group reads by UMI (GROUPREADSBYUMIDUPLEX). The GroupReadsByUMI command from fgbio groups aligned reads that share the same duplex consensus tag and genomic coordinates. These are assumed to belong to the same initial unique molecule, and are labelled as such, making a distinction between the reads that come from the forward or reverse strand of the initial molecule (labelled as A or B). PCR duplicates originating from each DNA strand of a unique DNA fragment are first grouped together, and then the two groups (corresponding to the two strands of a unique DNA fragment captured by the library, if available) are brought together and labelled under the same fragment number.

    The requirements for a set of reads to be part of the same group is a minimum mapping quality of 10 for all member reads and, at most, 1 nucleotide difference in the duplex tag sequences of the different member reads (--edits 1 --min-map-q 10). The number of edits should be adjusted accordingly based on the diversity of duplex tags used in the experimental protocol.

```
fgbio \
    -Xmx8g \
    --tmp-dir=. \
    GroupReadsByUmi \
    --edits 1     --min-map-q 10 \
    --strategy Paired \
    --input <sample>.resorted.bam \
    --output <sample>_umi-grouped.bam \
    --family-size-histogram <sample>_umi_histogram.txt
```

11. Collect duplex sequencing metrics. This is a two step process and these are computed in two different modes, leading to 4 different steps:
    a. COLLECTDUPLEXSEQMETRICS: using an fgbio tool generates information on multiple features of the dataset. Example 1: quantifying which is the distribution of the number of times each unique molecule was sequenced with strand

resolution. Example 2: Provides a count for the number of times each single UMI or combination of 2 UMIs was observed in the provided sample.

```
fgbio \
    -Xmx16g \
    --tmp-dir=. \
    --async-io=true \
    --compression=1 \
    CollectDuplexSeqMetrics \
    --input <sample>_umi-grouped.bam \
    --output <sample>.duplex_seq_metrics \
    --duplex-umi-counts=true;
```

b. FAMILYMETRICS: using as input the file mentioned in Example 1 of the step 11a above, generates metrics of the process of building consensus reads including plotting the distribution of single strand families that visually informs about the sequencing efficiency.

```
family_size_plots.py \
            --sample-name <sample> \
            --input-file <sample>.duplex_seq_metrics.duplex_family_sizes.txt \
            --output-file <sample>.family_sizes_plot_n_stats \
            --confidence-level '4 2 2'
```

c. COLLECTDUPLEXSEQMETRICSONTARGET: same as described above, but using only the molecules that are within the targeted regions.

d. FAMILYMETRICSONTARGET: same as before but with the output of step 11c.

12. Call duplex consensus reads (CALLDUPLEXCONSENSUSREADS). This step is responsible for collapsing all the reads that belong to the same original molecule into a single copy of that original molecule using the majority vote of all the copies that were sequenced. For this, it takes the groups of reads generated in the previous step and applies a minimal filter of base quality >= 20. All the initial molecules, even those with a single read supporting them, are part of the unmapped BAM file that is generated as output of this step.

```
fgbio \
    -Xmx3g \
    --tmp-dir=. \
    --async-io=true \
    --compression=1 \
    CallDuplexConsensusReads \
    --input <sample>_umi-grouped.bam \
    --output <sample>.consensus.bam \
    --threads 8 \
    --read-name-prefix <sample> \
    --read-group-id <sample> \
    --min-reads 1 1 0 --min-input-base-quality 20 --consensus-call-overlapping-bases true
```

13. Align consensus reads (ALIGNDUPLEXCONSENSUSBAM). This step realigns the consensus reads of all the unique molecules sequenced back to the same reference genome as before.

```
samtools fastq  <sample>.consensus.bam \
```

```
    | bwa mem -5 -L 1,1 -K 100000000 -t 32 -p -Y $FASTA - \
    | fgbio -Xmx4g \
        --compression 1 \
        --async-io=true \
        ZipperBams \
        --input /dev/stdin \
        --unmapped <sample>.consensus.bam \
        --ref $FASTA \
        --output <sample>.mapped.bam \
        --tags-to-reverse Consensus \
        --tags-to-revcomp Consensus \
        ;
```

14. Sort file to prepare it for downstream steps. (SORTBAMALLMOLECULES)

```
samtools sort  --write-index -@ 16 -o <sample>.sorted.bam -T tmp/ <sample>.mapped.bam
```

15. Filter reads with ambiguous mappability (ASMINUSXSDUPLEX). This step filters out reads that are mapping ambiguously to multiple positions of the genome. Specifically, the pipeline uses the difference between the score of the read's best alignment (AS) and the score of its second best alignment to any other position in the genome (XS). If for the two reads of a pair this difference is smaller than 50 units, the read pair is discarded. If one of the members of the pair shows an AS-XS smaller than 50, while the other shows AS-XS >= 50, both reads are kept. And if both reads show a bigger difference they are also kept.

```
as_minus_xs.py --input_bam <sample>.sorted.bam \
       --output_bam <sample>.filtered.AS-XS_50.bam \
       --output_bam_discarded <sample>.discarded_AS-XS_50.bam \
       --threshold 50 \
       --tthreads 2
```

16. Remove remaining non properly paired (SAMTOOLSFILTERALLMOLECULES) After filtering for mapping uniqueness, the pipeline only keeps the primary alignment of properly paired reads for downstream calling (samtools view <bam> -b -h --require-flags 0x2 --exclude-flags 0x900). Read pairs with one of the members incorrectly mapped are discarded.

    The resulting BAM file contains all consensus reads formed as explained above including those which do not form duplex reads (as explained below).

```
samtools sort \
       --threads 15 \
       -n \
       <sample>.filtered.AS-XS_50.bam \
       | samtools fixmate \
              --threads 15 \
              -r - <sample>.tmp.bam

samtools view --threads 15 <sample>.tmp.bam \
              -b -h --require-flags 0x2 \
              --exclude-flags 0x900 > <sample>.filtered.AS-XS_50.0x2.bam
```

Phase 3: All molecules files quality filtering

17. This BAM file with all the consensus reads formed contains both, those that made it into duplex and those that stayed as SSCs. Here we take the previous output and sort it (SORTBAMAMFILTERED) so that it can be reused for other processes or even outside of deepUMIcaller. We generally refer to this BAM file as the BAM file containing all molecules or ***BAM_unique***. This BAM file is part of the deepUMIcaller output.

```
samtools sort  -n -@ 16 \
       -o <sample>.sorted.bam -T tmp/ \
       <sample>.filtered.AS-XS_50.0x2.bam
```

18. The ASMINUSXS filtering step (step 15) not only provides the BAM file with the reads passing the mappability threshold but it also generates a BAM file with the reads that do not pass this threshold. Here we run two steps that used this BAM file with discarded reads to compute which areas of the targeted region seem to have mappability issues (DISCARDEDCOVERAGETARGETED), and also which other areas of the exome or a broader representation of the genome received reads with ambiguous mappability (DISCARDEDCOVERAGEGLOBAL).

```
bedtools \
    coverage \
    -mean \
    -a xgen-exome-hyb-panel-v2-targets-hg38.bed \
    -b <sample>.discarded_AS-XS_50.bam \
    > <sample>.bed
```

19. When parent_dna provided, samples with the same parent_dna are merged into a single BAM file to increase the depth of the final sample for analysis.
    a. MERGEBAMS. Combines the two or more all molecules BAM files into a single one preserving the same “properties” or status of the reads, since each consensus read at this point has been build independently for each library, merging the BAM files here ensures the full representativity of all the original unique molecules.
    b. SORTBAMMERGED. Sort the resulting BAM files accordingly, and the BAM files coming out of this step will be added to the same channel (see Nextflow channel explanation) coming from the SORTBAMAMFILTERED step (step 17)

At this point the pipeline has a small branch where the BAM unique files generated in steps 17 (and 19b) are passed both to steps 20 and 21 at the same time to undergo different stringency filters.

20. Here the BAM_unique files undergo a very permissive filter (FILTERCONSENSUSREADSAM) that only removes those molecules that did not undergo any minimal consensus filtering using FilterConsensus fgbio tool. This is filtered by applying a 2 1 0 quality threshold (see fgbio documentation for more information)

```
fgbio \
    -Xmx16g \
    --tmp-dir=. \
    --compression=1 \
```

```
    FilterConsensusReads \
    --input <sample>.sorted.bam \
    --ref GCA_000001405.15_GRCh38_no_alt_analysis_set.masked.fna \
    --output <sample>.filtered.bam \
    --min-reads 2 1 0  --min-base-quality 20 \
    --max-base-error-rate 0.1 0.1 0.1 --max-no-call-fraction 0.2 \
    --require-single-strand-agreement false;
```

a. To have this file in a usable format for downstream analysis we sort it appropriately. (SORTBAMAMCLEAN)

```
samtools sort  --write-index -@ 16 -o <sample>.sorted.bam -T tmp/ <sample>.filtered.bam
```

b. Adding one more QC at this step allows us to record some quality and quantity metrics of the process, using the QualiMap BAM QC tool (QUALIMAPQCALLMOLECULES). See below for a deeper explanation on the outputs.

```
qualimap \
    --java-mem-size=163840M \
    bamqc \
    --paint-chromosome-limits --genome-gc-distr HUMAN --skip-dup-mode 0 -outformat HTML \
    -bam <sample>.sorted.bam \
    --gff PanCancerPanel.original.hg38.chr.expanded250.bed6.bed \
    -p non-strand-specific \
    --collect-overlap-pairs \
    -outdir <sample>.duplex \
    -nt 2
```

21. At this point is where the molecules that do not meet the duplex quality standard requested are filtered out (FILTERCONSENSUSREADSDUPLEX). This filter is applied using the FilterConsensus fgbio tool and generally we apply a minimum support threshold of 4 2 2, and also force the error rate of each given base to be at most 0.1 meaning that bases where the majority vote is not unanimous require a disagreement ratio smaller than 1/10. This BAM file is referred to as BAM_filtered.

```
fgbio \
    -Xmx16g \
    --tmp-dir=. \
    --compression=1 \
    FilterConsensusReads \
    --input <sample>.sorted.bam \
    --ref GCA_000001405.15_GRCh38_no_alt_analysis_set.masked.fna \
    --output <sample>.filtered.bam \
    --min-reads 4 2 2     --min-base-quality 30   \
    --max-base-error-rate 0.1 0.1 0.1     --max-no-call-fraction 0.2 \
    --require-single-strand-agreement true;
```

22. To avoid overlapping read pair ends (insert size <300) interfering with the counting of mutations and depth these are hard clipped using ClipBam (CLIPBAM).

```
samtools sort -n -@ 16 -u <sample>.filtered.bam  | \
 fgbio \
        -Xmx32g \
        --tmp-dir=. \
        ClipBam \
        --input /dev/stdin/ \
```

```
        --ref GCA_000001405.15_GRCh38_no_alt_analysis_set.masked.fna \
        --clipping-mode Hard --clip-overlapping-reads true --clip-bases-past-mate true \
        --upgrade-clipping true --auto-clip-attributes true \
        --output <sample>.clipped.bam
```

23. With all the consensus building, filtering and read postprocessing steps completed, the duplex quality reads have been obtained and sorting them at this stage allows the usability inside and outside the pipeline (SORTBAMDUPLEXCONS). This is a critical output of the pipeline since downstream analysis via deepCSA requires this file as input.

```
samtools sort  --write-index -@ 16 -o <sample>.sorted.bam -T tmp/ <sample>.clipped.bam
```

24. The previous step generates the final processed BAM file and as the first step downstream, the basic QC metrics are collected using QualiMap BAM QC. (QUALIMAPQCDUPLEX)

```
qualimap \
    --java-mem-size=163840M \
    bamqc \
    --paint-chromosome-limits --genome-gc-distr HUMAN --skip-dup-mode 0 -outformat HTML \
    -bam <sample>.sorted.bam \
    --gff PanCancerPanel.original.hg38.chr.expanded250.bed6.bed \
    -p non-strand-specific \
    --collect-overlap-pairs \
    -outdir <sample>.duplex \
    -nt 2
```

25. An additional interesting QC step deepUMIcaller computes the coverage in a set of regions provided by the user (COVERAGEGLOBAL). These regions default to the same regions provided as target_bed_file but can be defined to any other BED file.

```
bedtools \
    coverage \
    -mean \
    -a xgen-exome-hyb-panel-v2-targets-hg38.bed \
    -b <sample>.sorted.bam \
    > <sample>.bed
```

26. All the QC metrics related to the duplex consensus building steps have been computed at this point of the execution. This (MULTIQCDUPLEX) is a first compilation of all of them to be able to merge these with the metrics of previous runs in case the complete run does not finish because of errors downstream and the final multiqc step cannot be executed.

```
multiqc -f .
```

Phase 4: Mutations: variant calling preparation and calling

27. Before proceeding to variant calling, we compute the depth of the duplex quality BAM file (COMPUTEDEPTH). The aim is to know which are the regions in which we have coverage so we could potentially detect mutations there. This is done using samtools depth.

```
samtools \
    depth \
    --threads 1 \
```

```
    -o <sample>.tsv \
    <sample>.sorted.bam
```

28. This step takes as input the samtools depth output to create a BED file with the regions that are covered by at least one read (CREATEBED). This is done using bedtools merge and the output defines which regions will be used for variant calling.

```
tail -n +2 <sample>.tsv | awk -F'\t' '{print $1, $2, $2}' OFS='\t' > <sample>.positions.tsv

bedtools merge -i <sample>.positions.tsv \
 -d 20 \
> <sample>.sequenced.bed;
```

29. With the duplex quality BAM file (step 26) and the definition of which positions might be mutated for each given sample, deepUMIcaller can proceed to the variant calling steps. This is handled by a subworkflow (BAM_CALL_VARDICT_PARALLEL) that does the variant calling in three steps.
    a. The first step (SPLIT_BED) splits the regions to focus on as many regions as requested by the user via param: vardict_chunks.

```
# Split the targets file into N chunks
total_lines=$(wc -l < <sample>.sequenced.bed)

# Calculate the number of lines per chunk
lines_per_chunk=$(( (total_lines + 2 - 1) / 2 ))

# Split the file into chunks with the calculated number of lines
# Using sample-specific prefix to avoid confusion in work directories
split -l ${lines_per_chunk} <sample>.sequenced.bed <sample>_chunk_
```

    b. The second step (CALLING_VARDICT_CHUNK) does the actual variant calling in parallel for each sample and piece of regions. This is done using VarDict-java v1.8.3 in the pileup mode to report all possible mutations in the desired regions. This execution yields all variants (SNVs and indels up to 100bp) in each sample. The output is filtered with the recommended VarDict filters, and only mutated positions were kept in the final VCF file.

```
echo "Running vardict-java on chunk <sample>_chunk_aa..."

# Run vardict-java on this chunk
vardict-java -G GCA_000001405.15_GRCh38_no_alt_analysis_set.masked.fna \
       -N <sample> -b <sample>.sorted.bam \
       -c 1 -S 2 -E 3 -g 4 -f 0.0 -r 1 -m 9999 -P 0 -p -z 1 -o 0.5 -L 100 \
       -th 32  <sample>_chunk_aa > <sample>_chunk_aa.raw.tsv

echo "Vardict finished for chunk <sample>_chunk_aa. Running teststrandbias..."

# Process with teststrandbias or awk
if false; then
    cat <sample>_chunk_aa.raw.tsv | teststrandbias.R | \
                var2vcf_valid.pl -N <sample> -A -E -f 0.0 -p 10 -m 20 \
                -v 1 > <sample>_chunk_aa.genome.vcf
else
    awk 'BEGIN { FS=OFS="\t" } {
        if (NF < 34) {
            print "ERROR: Unexpected column count " NF > "/dev/stderr";
```

```
            exit 1;
        }
        for (i = 1; i <= 20; i++) printf "%s\t", $i;
        printf "1.0\t1.0\t";
        for (i = 21; i <= NF; i++) {
            printf "%s", $i;
            if (i < NF) printf "\t";
        }
        printf "\n";
    }' <sample>_chunk_aa.raw.tsv | var2vcf_valid.pl   \
               -N <sample> -A -E -f 0.0 -p 0 -m 20 -v 2 \
                > <sample>_chunk_aa.genome.vcf
fi

echo "Done processing chunk <sample>_chunk_aa."
```

c. Since multiple variant calling steps will be executed for each starting sample (or duplex quality BAM file) we have to merge the different outputs belonging to the same sample. (MERGE_VARDICT_RESULTS)

```
echo "Merging VCF chunks..."

# Sort chunks to ensure consistent ordering (chunk_aa, chunk_ab, etc.)
vcf_files=($(ls *.genome.vcf | sort))

# Extract the header from the first VCF chunk
grep "^#" "${vcf_files[0]}" > <sample>.genome.vcf

# Concatenate all genome VCF chunks (excluding headers)
for vcf_file in "${vcf_files[@]}"; do
    grep -v "^#" "$vcf_file" >> <sample>.genome.vcf
done

echo "Done concatenating VCFs. Applying AWK filter..."

# Apply the AWK filter to create the final VCF
awk '$5!="."' <sample>.genome.vcf > <sample>.vcf

echo "Done. Compressing genome VCF..."

gzip <sample>.genome.vcf

echo "Merging RAW TSV chunks..."

# Merge raw TSV chunks (order by chunk name for determinism)
raw_files=( $(ls *.raw.tsv | sort) )
if [ ${#raw_files[@]} -gt 0 ]; then
    cat "${raw_files[@]}" > <sample>.raw.tsv
    gzip <sample>.raw.tsv
fi

echo "Done merging all results."
```

Phase 5: Mutations: postprocessing of variant calls

Up to this point, we generated the duplex quality reads and performed the main step of variant calling, the downstream steps consist of postprocessing of the variant calls to add as much

information as possible on a mutation specific basis so that it can be used for decision making in further downstream analysis outside of deepUMIcaller. Most of these postprocessing steps are within the RECOUNTMUTS subworkflow, and they include a combination of steps that prepare files for flagging or annotating mutations and the steps where the mutations are actually processed.

30. An initial step in this postprocessing is to prepare BED files with the information on the mutated positions so that these can be more easily compared to other reference or masking files (RECOUNTMUTS:READJUSTREGIONS.*)
    a. READJUSTREGIONS_AMP (the position of each mutation is amplified 2 bps on each side)

```
grep -v '#' <sample>.vcf | \
    awk '{ sum = length($4); print $1"\t"$2-1-2"\t"$2 -1 + sum + 2"\t"$1";"$2";"$4";"$5}' | \
    sort -k1,1 -k2,3n \
    > <sample>.duplex.vcf_derived.many.withID.bed

cut -f-3 <sample>.duplex.vcf_derived.many.withID.bed > <sample>.duplex.vcf_derived.many.bed;

bedtools \
    merge \
    -i <sample>.duplex.vcf_derived.many.bed \
    -d 10 \
    > <sample>.duplex.vcf_derived.bed;


rm <sample>.duplex.vcf_derived.many.bed;

bedtools \
    merge \
     -i <(cat <sample>.sequenced.bed <sample>.duplex.vcf_derived.bed | cut -f -3 | sort -k1,1
-k2,3n) \
    -d 10 \
     \
    > <sample>.duplex.regions_n_mutations.bed
```

    b. READJUSTREGIONS_NOAMP (only the position of each mutation proceeds to the final BED file)

```
grep -v '#' <sample>.vcf | \
    awk '{ sum = length($4); print $1"\t"$2-1-0"\t"$2 -1 + sum + 0"\t"$1";"$2";"$4";"$5}' | \
    sort -k1,1 -k2,3n \
    > <sample>.duplex.vcf_derived.many.withID.bed

cut -f-3 <sample>.duplex.vcf_derived.many.withID.bed > <sample>.duplex.vcf_derived.many.bed;

bedtools \
    merge \
    -i <sample>.duplex.vcf_derived.many.bed \
    -d 10 \
    > <sample>.duplex.vcf_derived.bed;

rm <sample>.duplex.vcf_derived.many.bed;

bedtools \
    merge \
```

```
     -i <(cat <sample>.sequenced.bed <sample>.duplex.vcf_derived.bed | cut -f -3 | sort -k1,1
-k2,3n) \
    -d 10 \
     \
    > <sample>.duplex.regions_n_mutations.bed
```

31. We take the duplex quality BAM file (step 23) and run samtools mpileup on it so that we have information on the nucleotide of all the reads covering each specific position in the covered regions of the genome (RECOUNTMUTS:PILEUPBAM). This will be used for recounting the reference and alternative reads supporting each mutation, in downstream steps.

```
samtools mpileup \
    --fasta-ref GCA_000001405.15_GRCh38_no_alt_analysis_set.masked.fna \
    --output <sample>.duplex.mpileup \
    --no-BAQ --max-depth 0 --min-BQ 2 --no-output-ends --output-extra QNAME \
    --positions <sample>.duplex.regions_n_mutations.bed \
    <sample>.sorted.bam
bgzip -@1 <sample>.duplex.mpileup
tabix -s 1 -b 2 -e 2 <sample>.duplex.mpileup.gz
```

32. As part of the duplex consensus building process, when there is a disagreement between more than 1 out of 10 PCR copies within a DSC, an N is placed at that position of the duplex consensus read, as a token of that disagreement. This step (RECOUNTMUTS:NSXPOSITION) collects the information on the number of Ns for all the positions processed by samtools mpileup (step 31). This will be used downstream for flagging mutations in positions with a high proportion of Ns .

```
zcat <sample>.duplex.mpileup.gz | \
                    awk                         'BEGIN{FS=OFS="\t"}                         {print
$1"\t"$2"\t"$4"\t"gsub(/N/,"",$5)"\t"gsub(/n/,"",$5)"\t"gsub(/\*/,"",$5)}' | \
        awk '{$3=$3-$6;$4=$4+$5; print $1"\t"$2"\t"$3"\t"$4}' | \
        bgzip -@ 1 \
        > <sample>.duplex.Ns_per_position.tsv.gz;
tabix -s 1 -b 2 -e 2 <sample>.duplex.Ns_per_position.tsv.gz;
```

33. We take the BAM unique file (step 20a) and run samtools mpileup on it so that we have information on the nucleotide of all the reads covering each specific position in the covered regions of the genome (RECOUNTMUTS:PILEUPBAMALL). In this case we do not do it in the same file that was used for mutation calling but in a file that contains several additional molecules with lower levels of consensus support. This will be used for recounting the reference and alternative reads supporting each mutation, in downstream steps,

```
samtools mpileup \
    --fasta-ref GCA_000001405.15_GRCh38_no_alt_analysis_set.masked.fna \
    --output <sample>.duplex.mpileup \
    --no-BAQ --max-depth 0 --min-BQ 2 --no-output-ends --output-extra QNAME \
    --positions <sample>.duplex.vcf_derived.bed \
    <sample>.sorted.bam
bgzip -@1 <sample>.duplex.mpileup
tabix -s 1 -b 2 -e 2 <sample>.duplex.mpileup.gz
```

34. Having the pileup information for all positions with coverage is useful in certain processes but increases the size and the difficulties with handling large dataframes. This step (RECOUNTMUTS:QUERYTABIX) uses tabix to query only the mutated positions and few neighbouring nucleotides

```
tabix <sample>.duplex.mpileup.gz -R <sample>.duplex.vcf_derived.bed | bgzip > <sample>.duplex.mutated_positions.tsv.gz
```

35. We use the mpileup output generated in step 31 and queried in to go through the mutations called by VarDictJava and provide an additional version of read counts to that available when doing the calling with VarDictJava. (RECOUNTMUTS:PATCHDP)

```
recompute_depth.py \
        --mpileup-file <sample>.duplex.mutated_positions.tsv.gz \
        --vcf-file <sample>.vcf \
        --output-filename <sample>.duplex.readjusted.vcf \
```

36. With the file generated in the previous step and that coming from step 33 we apply the same rationale but using the information from lower quality reads (RECOUNTMUTS:PATCHDPALL). Note that we only add information on top of mutations called with the duplex quality reads, no additional mutations are added here.

```
recompute_depth.py \
        --mpileup-file <sample>.duplex.mpileup.gz \
        --vcf-file <sample>.duplex.readjusted.vcf \
        --output-filename <sample>.duplex.AM.readjusted.vcf \
        --suffix AM
```

37. The following mutation postprocessing steps consist of using BED files with different information on genomic features to annotate the mutations. These are positions likely to accumulate mapping or calling artifacts. (RECOUNTMUTS filter panels)
    See an additional explanation and comparison between the masks for the human genome (hg38) here: https://github.com/bbglab/deepUMIcaller/blob/dev/docs/usage.md#filtering-bed-files
    a. FILTERLOWMAPPABLE: the pipeline labels as *low_mappability* all mutations overlapping ENCODE blacklisted regions, taken from the ENCFF356LFX dataset (https://www.nature.com/articles/s41598-019-45839-z#data-availability).

```
if [[ "GRCh38.encode.blacklist.bed" == *.gz ]]; then
    bedtools intersect -a <sample>.duplex.vcf_derived.many.withID.bed -b <(zcat GRCh38.encode.blacklist.bed) -u > <sample>.duplex.low_mappability_file.bed
else
     bedtools intersect -a <sample>.duplex.vcf_derived.many.withID.bed -b GRCh38.encode.blacklist.bed -u > <sample>.duplex.low_mappability_file.bed
fi

# If there is nothing in the intersection, just copy the VCF
if [ -s <sample>.duplex.low_mappability_file.bed ]; then
    add_filter_from_bed.py \
        <sample>.duplex.AM.readjusted.vcf \
        <sample>.duplex.low_mappability_file.bed \
        <sample>.duplex.low_mappability.vcf \
        low_mappability
else
    cp <sample>.duplex.AM.readjusted.vcf <sample>.duplex.low_mappability.vcf
```

```
fi
```

b. FILTERLOWCOMPLEX: Mutations falling in RepeatMasker labeled positions are labeled as *low_complexity*.
Same logic as above, different regions definition and label.

```
add_filter_from_bed.py \
        <sample>.duplex.low_mappability.vcf \
        <sample>.duplex.low_complex_file.bed \
        <sample>.duplex.low_complex.vcf \
        low_complex
```

c. FILTERNANOSEQSNP: Mutations that coincide with positions with known SNPs are labeled as nanoseq_snp. This file with an SNP mask is provided by the Nanoseq team (https://drive.google.com/drive/folders/1wqkgpRTuf4EUhqCGSLA4fIg9qEEw3ZcL).
Same logic as above, different regions definition and label.

```
  add_filter_from_bed.py \
        <sample>.duplex.low_complex.vcf \
        <sample>.duplex.nanoseq_snp_file.bed \
        <sample>.duplex.nanoseq_snp.vcf \
        nanoseq_snp
```

d. FILTERNANOSEQNOISE: Mutations that coincide with positions known to be error prone as classified by the Nanoseq team are labeled as nanoseq_noise. This file with the NOISE mask is provided by the Nanoseq team here: https://drive.google.com/drive/folders/1wqkgpRTuf4EUhqCGSLA4fIg9qEEw3ZcL.
Same logic as above, different regions definition and label.

```
add_filter_from_bed.py \
        <sample>.duplex.nanoseq_snp.vcf \
        <sample>.duplex.nanoseq_noise_file.bed \
        <sample>.duplex.nanoseq_noise.vcf \
        nanoseq_noise
```

38. Using the information on the overall distribution of the number of Ns per position and the count of Ns for each mutation, this step (RECOUNTMUTS:FILTERNRICH) is responsible for flagging those mutations with an unexpectedly high proportion of Ns[9]. The output of this step is the final VCF file with all the mutations and all the annotated flags. This will be stored as part of the output in the mutations_vcf file.

```
add_filter_nrich.py \
        --vcf_file <sample>.duplex.nanoseq_noise.vcf \
        --ns_position_file <sample>.duplex.Ns_per_position.tsv.gz \
        --output_filename <sample>.duplex.filtered.vcf \
        --filter_name n_rich \
        --min_valid_depth 25
```

Phase 6: Mutations QC

The remaining steps downstream of the generation of this VCF file are QCs of the mutations and a final step to compile the information from most of the metrics computed in the process.

39. This filtering step (RECOUNTMUTS:FILTERVCFSOMATIC) removes mutations flagged with some of the likely artifactual masks and applies a VAF threshold of 0.35 to keep only the likely somatic mutations for the generation of the mutational profile plots downstream.

```
filtervcf.py \
            <sample>.duplex \
            <sample>.duplex.filtered.vcf \
            no_pileup_support,low_complex_repetitive,low_mappability,n_rich \
            0.35 \
            <sample>.duplex \
            True
```

40. Taking the output generated by step 39, here deepUMIcaller calls three different times the SigProfilerMatrixGenerator tool to obtain a mutational profile plot for each of the samples included in the run using either all mutations, only purine centric mutations and only pyrimidine centric mutations. The code is the same for the three different processes, and the only difference resides in the input files.

```
mkdir input_mutations
cp *.vcf input_mutations/.

SigProfilerMatrixGenerator matrix_generator \
            cord_blood_tests \
            GRCh38 \
            input_mutations/ \
            --plot --tsb_stat --seqInfo --cushion 100
```

    a. SIGPROFPLOT
    b. SIGPROFPLOTPUR
    c. SIGPROFPLOTPYR

41. A different filter of mutations is required for producing the plot that informs us on the distribution of mutations along the positions of the read. The main difference between this step (RECOUNTMUTS:FILTERVCFPLOT) and the previous filtering step (39) is that here only mutations that have been observed a single time are kept. If there had been errors incorporated in the process of the library prep, these errors would be randomly distributed so it is unlikely that any mutation occurring more than once is an artifact introduced during fragmentation or ligation steps.

```
filtervcf.py \
          <sample>.duplex \
          <sample>.duplex.filtered.vcf \
          no_pileup_support,low_complex_repetitive,low_mappability,n_rich,singletons_only \
          0.35 \
          <sample>.duplex
```

42. Using the filtered mutations (step 41) and the duplex quality BAM file (step 23) RECOUNTMUTS:MUTSPERPOS step computes the number of SNVs that have been detected in each position along the sequence of the read. These values are computed for each of the 6 possible nucleotide changes, and the results are represented in a plot such as those in Fig. 4.

```
grep -v '##' <sample>.duplex.filter_mutations.vcf > <sample>.duplex.no_header.vcf;
count_muts_per_cycle.py \
            --inFile <sample>.sorted.bam \
            --inVCF <sample>.duplex.no_header.vcf \
            -o <sample>.duplex \
            -l 142  --filter INCLUDE  -t 0  --clonality_limit 0.1  -c
```

43. In addition to the plots, step 42 provides a table with the counts of mutations per position in the read, and this specific table for all of the samples in the cohort that is being run is compiled by the RECOUNTMUTS:COHORTMUTSPERPOS step that takes care of summarizing the presence or absence of a potential enrichment of mutations at the beginning of the reads.

```
summarize_muts_per_cycle.py --initial 5
```

44. CUSTOM_DUMPSOFTWAREVERSIONS compiles the versions of all the tools used in deepUMIcaller.

45. MULTIQC

```
multiqc -f .
```

Main outputs

The main outputs of deepUMIcaller are:

- Mutations file after the application of all filters (generated at step 38)
- BAM file with duplex molecules (generated at step 23)
- Quality metrics computed along the process (most of them compiled at step 45 + those at steps 11b and 11d)

Versions of the variant allele frequency (VAF) computation

The four values of VAF for each mutation are calculated:

- **VAF_vd**: this is the VAF computed by VarDict at the time of calling the variants with its internal decision making on which reads to count for the reference/alt/total.
- **VAF**: The quotient of duplex consensus reads (at a given quality filter) supporting the alternate allele and all duplex consensus reads covering the mutated position. Computed at step 35.
- **VAF_AM**: also referred to as the all molecules VAF. This is the quotient of all consensus (duplex or single-strand) reads supporting the alternate allele and all consensus reads covering the mutated position. Computed at step 36.
- **VAF_ND**: also known as non-duplex VAF. Is the quotient between all the non duplex reads present in the all molecules file supporting the alternate allele and the all molecules non duplex reads covering the mutated position.

Family metrics

Steps 11b and 11d output a table with several metrics of the duplex building process as well as the family metrics plots presented in Fig. 3a-d panels.

QualiMap BAM QC outputs

This tool is run at three different levels: raw reads, all unique molecules and duplex quality reads.

In each of these cases for each sample we obtain the following outputs:

<sample>.<raw|all_molecules|duplex>/

```
├── qualimapReport.html          Main HTML report
├── genome_results.txt           Summary statistics
├── raw_data_qualimapReport/     Detailed metrics
│   ├── coverage_histogram.txt
│   ├── mapped_reads_gc-content_distribution.txt
│   └── ...
└── images_qualimapReport/       Quality plots
    ├── coverage_across_reference.png
    ├── insert_size_histogram.png
    └── ...
```

Critical metrics reported:

- Coverage statistics: Mean, median, SD across target regions
- Mapping quality: Distribution of MAPQ scores
- Insert size distribution: For properly paired reads
- GC content: Bias assessment
- Duplication rate: PCR and optical duplicates
- On-target rate: Percentage of reads in target regions

Checking for artifacts

As some sequencing artifacts related to DNA damage and end-repair are known to affect the ends of the reads more frequently, you can check the distribution of all mutations along duplex consensus reads to check for potential end-of-reads artifacts (Figure 4).

If there is an increased proportion of mutations in the first bps of the reads (Fig. 4b,c,d) you should re-run the samples clipping a bigger number of bps at the beginning of the reads. A certain amount of these mutations is expected, particularly for experimental protocols that include an end-repair step.

A simple way to check for enrichment at the beginning of the reads is by checking the cohortmutsperpos outputs and in particular those cells with values higher than 2.

Additional options

In addition to this canonical procedure listed above, deepUMIcaller has the flexibility to accommodate different parallelization strategies as well as different inputs.

Implemented parallelization strategies:

- Input reads for a sample provided splitted across several FASTQs and splitted_original_sample parameter set to true.

    - Steps 3-8 can be run independently for each batch of reads. This allows the parallel processing of more reads of the same sample at a time speeding up the overall pipeline execution.
    - This functionality is triggered automatically when there are multiple rows in the input.csv file with the same sample_name, and it then requires an additional step to merge the BAM files coming from the same sample before Step 9.
        - MERGEBAM
- Split reads aligned to different chromosomes.
    - Steps 9-14 can be run independently for each chromosome. This also allows the parallel processing of more reads of the same samples at a time speeding up pipeline execution specially for big libraries.
    - This is triggered by split_by_chrom and requires specific additional steps to be executed.

Implemented deepUMIcaller starting points:

- mapping (default)
- groupreadsbyumi
- unmapped_consensus
- allmoleculesfile
- filterconsensus
- calling

More information on these and other additional options can be found at: https://github.com/bbglab/deepUMIcaller

Mutation filters

A range of filters are applied to the mutations identified by deepUMIcaller. The philosophy of the pipeline, as is the case with most of its kind, is to add flags to called mutations, so as to provide the user with information to make their own decisions on which mutations to filter. While some of these filters are calculated –and added as mutation flags– by deepUMIcaller (described above), others depend on the set of samples analyzed as a cohort and are thus computed by deepCSA. Below, we provide a list of the filters that are annotated and applied to obtain a high quality set of somatic mutations in a cohort.

Mutation filters can be grouped in two steps. The first group includes criteria that label mutations that are more likely to be artifacts or outside of the main regions of interest, the second group flags mutations that are likely to be germline. Filters from each of these groups are used in steps X and Y of deepCSA respectively.

Group 1:

1. Variants in positions flagged as *N-rich* in the sample in question or in more than 10% of the samples of the cohort, (deepUMIcaller, deepCSA)

2. Variants in reads that contain a high proportion of *N*s. (deepUMIcaller)
3. Variants not supported by the reanalysis of the variant calling with samtools mpileup described above, (deepUMIcaller)
4. Variants in low mappability areas, (deepUMIcaller)
5. Variants outside the well covered areas (outside of the consensus panel), (deepCSA)
6. Variants with duplex depth below 100, (deepCSA)
7. Variants with VAF_AM value that is 3 times or more its duplex VAF (defined based on exploration of Supplementary Note Fig. 3.6), (deepCSA)
8. (human-specific) Variants overlapping Nanoseq NOISE mask, (deepUMIcaller | deepCSA)

Group 2:

9. (human-specific) Variants overlapping Nanoseq SNP mask, (deepUMIcaller | deepCSA)
10. Variants labelled as gnomAD_SNPs, these are variants with a gnomAD allele frequency > 0.1 % (source Ensembl 111), (deepCSA)
11. Variants that have been detected as likely SNPs in individuals of the same cohort, (deepCSA)
12. Variants with any of the three VAF metrics (VAF, VAF_AM, vd_VAF) equal to or greater than 0.3 (likely germline). (deepCSA)

Cross-contamination analysis

DeepCSA also performs cross-contamination analysis to assess if the DNA of a sample could be contaminated with the DNA of another. This could be detected by the presence of the SNPs of a sample as somatic mutations in another. This assessment is automatically performed by deepCSA in step 29, and the outputs are provided in the contamination directory.

Extended protocol for deepCSA

This protocol describes the step-by-step execution of the deepCSA (deep Clonal Structure Analysis) pipeline that implements mutagenesis and selection analyses from the mutations identified (using deepUMIcaller or any other DNA duplex mutation caller) across a cohort of samples. The analyses integrated in deepCSA include variant annotation, mutational signature discovery, and the detection of positive selection using an array of methods. Here, we describe the computational workflow and all the mandatory steps of deepCSA.

Scope

This extended protocol covers the complete computational workflow for analyzing bladder cancer samples, including:

- Quality control and variant annotation
- Creation of sample-specific genomic panels
- Mutational profile and signature analysis

- Detection of positive selection (dN/dS, OncodriveFML, OncodriveCLUSTL, Oncodrive3D)
- Omega (ω) selection coefficient estimation
- Mutation density and rate calculations

Documented at commit: 5c252a0518e6c33bf32970184d617f58a307697a, run with Nextflow version: 25.04.3, execution profile: singularity, irbcluster and reference genome: GRCh38.

Requirements

Computational:

- High-performance computing cluster
- Maximum resources per task:
    - CPUs: 56 cores
    - Memory: 256 GB
    - Time: 240 hours
- Total amount of storage used by temporary directories: 25 GB
- Container Technology: Singularity/Apptainer enabled

Input data:

- Sample sheet (CSV format): Contains sample identifiers, BAM file paths, and sequencing metadata.
- Reference Genome
    - File: GCA_000001405.15_GRCh38_no_alt_analysis_set.masked.fna
    - Assembly: GRCh38
- VEP Cache (directory cache)
    - Version: 111
    - Species: Homo sapiens or Mus musculus
- Annotation Resources
    - COSMIC reference signatures: COSMIC_v3.4_SBS_GRCh38.txt
    - CADD scores: v1.7 for GRCh38
    - Oncodrive3D datasets and annotations (version 240506)
- Custom Annotations
    - TERT promoter regions: TERT_regions.tsv
    - Hotspot definitions: all_intogen_v2024.header.tsv
    - Protein domains: bbgdomains.v2025.annotated_deepCSA.tsv
- Filtering Resources
    - NanoSeq SNP mask: SNP_GRCh38.wgns.bed.gz
    - NanoSeq noise regions: NOISE_GRCh38.wgns.bed.gz
    - ENCODE blacklist: GRCh38.encode.blacklist.bed
    - Low complexity regions: hg38_lowcomplexity_repetitive_regions.mutcoords.bed
- Sample Metadata
    - Features table: metadata_bladder.deepcsa_input.tsv
    - Contains clinical variables: SEX, BLADDER_LOCATION, HISTORY_OF_SMOKING

Procedure

Phase 1: Input Validation and Setup

1. Validates input CSV format and checks for required columns. (INPUT_CHECK:SAMPLESHEET_CHECK)

```
check_samplesheet.py \
    deepCSA_input.med.csv \
    samplesheet.valid.csv
```

2. This step generates all the groups of samples that will then undergo the analysis. This will always preserve the samples individually and the entire group of samples together, and additionally it might contain separate custom groups defined by the information in the features_table parameters section (TABLE2GROUP). Feature table processing. Parses clinical metadata and creates sample groupings based on specified variables.

```
features_1table2groups.py \
    --table-filename metadata_bladder.deepcsa_input.tsv \
    --separator tab \
    --unique-identifier SAMPLE_ID \
    --groups "[ [SEX], [BLADDER_LOCATION], [BLADDER_LOCATION, SEX], [HISTORY_OF_SMOKING] ]"
```

Phase 2: Reference panel generation: sequencing depth analysis, panel generation, consensus panel definition, definition of subgenic regions & final processing of the depths.

3. Compute sequencing depth (DEPTHANALYSIS:COMPUTEDEPTHS). Calculates read depth at each genomic position across all samples (input BAM files). It also optionally filters positions to those with mean depth bigger or equal to a predefined value. ≥30× in the example below (configured via use_custom_minimum_depth param).

```
ls -1 *.bam > bam_files_list.txt

samtools depth -H -@ 2 -f bam_files_list.txt | \
    tail -c +2 | \
    awk 'NR == 1 {print; next}
         {sum = 0;
          for (i=3; i<=NF; i++) sum += $i;
          mean = sum / (NF - 2);
          if (mean >= 30) print}' | \
    gzip -c > all_samples.depths.tsv.gz
```

Once the sequencing depth per position has been obtained a set of methods within the CREATEPANELS subworkflow are responsible for generating reference files that will indicate in which regions to focus the analysis.

4. Extract genomic positions from sequencing depth per position (SITESFROMPOSITIONS). Converts depth-passing positions to all possible mutations in those genomic sites. This is prepared in the format required for VEP annotation.

```
cat <(printf "CHROM\tPOS\n") \
    <(zcat all_samples.depths.tsv.gz | cut -f1,2 | sed 's/^chr//g' | awk 'length($1) <= 2') \
    > captured_positions.tsv
```

```
sites_table_from_positions.py \
    --input-positions captured_positions.tsv \
    --genome-assembly hg38 \
    --output-file-with-sites captured_positions.sites4VEP.tmp.tsv

awk '{print "chr"$0}' captured_positions.sites4VEP.tmp.tsv > captured_positions.sites4VEP.tsv
```

5. VEP annotation of saturated panel positions (VCFANNOTATEPANEL:ENSEMBLVEP_VEP). Annotates all captured genomic positions with variant consequences and gene information.

```
vep \
    -i captured_positions.sites4VEP.tsv \
    -o captured_panel.tab.gz \
    --no_stats --cache --offline --symbol --protein --canonical --tab \
    --compress_output bgzip \
    --fasta GCA_000001405.15_GRCh38_no_alt_analysis_set.masked.fna \
    --assembly GRCh38 \
    --species homo_sapiens \
    --cache_version 111 \
    --dir_cache ${PWD}/vep \
    --fork 50
```

6. Post-Process VEP Panel Annotation (POSTPROCESSVEPPANEL). Ensembl VEP provides multiple annotations per each mutation. In this step, a single annotation per position-change is selected and the script extracts the relevant VEP fields and provides two different files with distinct sets of annotations, one more complete than the other: simple & rich (simple + protein position and amino acid change).

```
zegrep -v '^##' captured_panel.tab.gz | \
    cut -f 1,5,7,10,11,16,18,21-22 | \
    awk '$7!="-"' | \
    uniq | \
    tail -n +2 | \
     awk -F'\t' 'BEGIN {OFS = "\t"} {split($1, a, "[_/]"); print a[1], a[2], a[3], a[4], $1,
$2, $3, $4, $5, $6, $7, $8, $9}' | \
    gzip > captured_panel.tmp.gz

panel_postprocessing_annotation.py \
    --vep_output_file captured_panel.tmp.gz \
    --assembly hg38 \
    --output_file captured_panel.tab.compact \
    --only_canonical
```

7. Apply custom annotations for specific areas of the panel that should be treated differently such as the case of the TERT promoter. This script integrates custom region definitions (e.g., TERT promoter), while keeping the stored information as it was.
    a. Simple: (CUSTOMPROCESSING)

```
panel_custom_processing.py \
    --vep-output-file captured_panel.tab.compact.tsv \
    --custom-regions-file TERT_regions.tsv \
    --customized-output-annotation-file captured_panel.tab.compact.custom.tsv \
    --simple
```

b. Rich: (CUSTOMPROCESSINGRICH)

```
panel_custom_processing.py \
    --vep-output-file captured_panel.tab.compact_rich.tsv \
    --custom-regions-file TERT_regions.tsv \
    --customized-output-annotation-file captured_panel.tab.compact_rich.custom.tsv
```

8. Use information from a predefined set of protein domains or relevant subgenic regions and map those to your regions of interest (DOMAINANNOTATION). Convert protein domain boundaries to genomic positions.

```
panels_dna2domain.py \
    --consensus-panel-rich captured_panel.tab.compact_rich.custom.tsv \
    --domains bbgdomains.v2025.annotated_deepCSA.tsv \
    --output seq_panel.domains.bed4.bed
```

9. Create captured panel versions (CREATECAPTUREDPANELS). From a single broad panel create several versions of it only according to its consequence type. This will generate both a TSV and a BED version for each of the multiple panel versions (all, protein-affecting, non-protein-affecting, exons, synonymous).

```
create_panel_versions.py \
    --compact-annot-panel-path captured_panel.tab.compact.custom.tsv \
    --output captured_panel;

for captured_panel in $(ls -l *.tsv | grep -v '^l' | awk '{print $NF}'); do
    bedtools merge \
        -i <(
            tail -n +2 $captured_panel | \
            awk -F'\t' '{print $1, $2-1, $2}' OFS='\t' | uniq
        ) > ${captured_panel%.tsv}.bed;
done
```

10. Given each of the different capture panels, a predefined depth (enough to trust all the analysis done at that depth) and consensus_compliance (proportion of samples that must meet the depth requirement for that position to stay in the consensus panel) thresholds. deepCSA creates the consensus panels. “consensus_all” contains the maximum range of regions that are properly sequenced, while for the rest of the panels the user can provide a list of the genes (params.subset_genes) to focus on, and run the methods only for those.
    a. CREATECONSENSUSPANELSALL
    b. CREATECONSENSUSPANELSPROTAFFECT
    c. CREATECONSENSUSPANELSNONPROTAFFECT
    d. CREATECONSENSUSPANELSEXONS
    e. CREATECONSENSUSPANELSINTRONS
    f. CREATECONSENSUSPANELSSYNONYMOUS

```
create_consensus_panel.py \
           --compact_annot_panel_path captured_panel.<panel_version>.tsv \
           --depths_path all_samples.depths.tsv.gz \
           --version <panel_version> \
           --consensus_min_depth 190 \
           --compliance_threshold 0.8 \
```

```
            --genes_subset <genes>

bedtools merge \
        -i <(
        tail -n +2 consensus.<panel_version>.tsv | \
        awk -F'\t' '{print $1, $2-1, $2}' OFS='\t' | uniq
        ) > consensus.<panel_version>.bed;
```

(optional) the user can request the generation of sample specific panels with well covered regions specifically for each sample, but these are not used in any way in deepCSA.

On top of the panels created so far, the user can define subgenic regions that will act as independent genomic elements in some downstream analysis, this is handled by the ENRICHPANELS subworkflow.

11. Create DNA-to-Protein Mapping Table (ENRICHPANELS:DNA2PROTEINMAPPING). Generates lookup table for converting genomic coordinates to protein positions with additional information on all the exons of the protein coding genes that are covered by the panels.

```
cut -f 1,2,6 consensus.exons_splice_sites.tsv | uniq > exons_splice_sites.panel.unique.tsv
panels_computedna2protein.py \
            --mutations-file all_samples.somatic.mutations.tsv \
            --consensus-file exons_splice_sites.panel.unique.tsv \
            --depths-file all_samples_indv.depths.tsv.gz
```

12. Create expanded regions for all possible panels so that we can compute all possible measurements of the mutation density as well.
    a. ENRICHPANELS:EXPANDREGIONSALL
    b. ENRICHPANELS:EXPANDREGIONSNONPROT
    c. ENRICHPANELS:EXPANDREGIONSPROT
    d. ENRICHPANELS:EXPANDREGIONSSYNONYMOUS
    e. ENRICHPANELS:EXPANDREGIONSEXONS

```
add_subgenicregions.py --panel_file consensus.all.tsv \
                  --autoexons panel_exons.bed4.bed \
                  --autodomains seq_panel.domains.bed4.bed
```

13. Generate annotated depth file for each sample or analysis group (ANNOTATEDEPTHS)

```
cut -f 1,2,9 captured_panel.all.tsv | uniq > captured_panel.all.tsv.contexts
merge_annotation_depths.py \
    --annotation captured_panel.all.tsv.contexts \
    --depths all_samples.depths.tsv.gz \
    --json_file all_groups.json \
    --input_csv deepCSA_input.med.csv
```

14. This step generates a summary of the depth per gene and per sample which are part of the critical QCs. Generate depth summary statistics available in depthssummary (PLOTDEPTHS*). This is repeated for panels: all consensus, exons consensus and exons. See: https://github.com/bbglab/deepCSA/blob/main/docs/output.md for an

explanation of the differences between the outputs of these three calls to the subworkflow.

a. PLOTDEPTHSALLCONS:
    i. SUBSETDEPTHS

```
TABIX-SUBSET-code run with the all consensus panel
(see SUBSET's code below)
```

    ii. ONCODRIVEFMLBED

```
sh createcustombed.sh captured_panel.exons_splice_sites.tsv oncodrivefml > exons_splice_sites.exons.annotated.bed
```

    iii. DEPTHSSUMMARY

```
plot_depths.py \
            --sample_name all_samples.all_cons \
            --depth_file all_samples.subset_depths.tsv.gz \
            --panel_bed6_file all.annotated.bed \
            --panel_name all \
            --plot_within_gene False;
```

b. PLOTDEPTHSEXONS:
    i. SUBSETDEPTHS

```
TABIX-SUBSET-code run with the exons panel
(see SUBSET's code below)
```

    ii. ONCODRIVEFMLBED
    iii. DEPTHSSUMMARY

c. PLOTDEPTHSEXONSCONS
    i. SUBSETDEPTHS

```
TABIX-SUBSET-code run with the exons consensus panel
(see SUBSET's code below)
```

    ii. ONCODRIVEFMLBED
    iii. DEPTHSSUMMARY

Phase 3: Mutation preprocessing: variant annotation, filter labelling, QCs and visualization and final variant selection

The mutation information coming from the VCF files of all samples is processed in the MUT_PREPROCESSING subworkflow.

15. VEP Annotation of Sample Variants (VCFANNOTATE:ENSEMBLVEP_VEP). Adds variant ID in format "chr:pos_ref>alt" for downstream tracking and annotates all variants detected in each sample VCF file in parallelly executed processes.

```
cat <(grep '#' <sample>.med.filtered.vcf) \
    <(grep -v '#' <sample>.med.filtered.vcf | \
      awk -F'\t' '{OFS="\t"; $3=$1":"$2"_"$4">"$5; print}') \
```

```
    > <sample>.med.filtered.vcf.4vep.vcf

vep \
    -i <sample>.med.filtered.vcf.4vep.vcf \
    -o <sample>.tab.gz \
    --no_stats --cache --offline --symbol --protein --canonical \
    --af_gnomadg --af_gnomade --tab \
    --compress_output bgzip \
    --fasta GCA_000001405.15_GRCh38_no_alt_analysis_set.masked.fna \
    --assembly GRCh38 \
    --species homo_sapiens \
    --cache_version 111 \
    --dir_cache ${PWD}/vep \
    --fork 2
```

16. Aggregate individual variant annotations to select a single consequence annotation for each mutation and apply it uniformly across all the other samples (SUMANNOTATION). At this stage we add information about some hotspot mutations and also apply some basic filters based on gnomAD allele frequency.

```
zcat <$(ls *.tab.gz | head -1) | grep '#' | grep -v '##' > header.tsv;
for file in *.tab.gz; do
        zgrep -v '#' $file >> all_samples.vep.tab.tmp;
        echo $file;
done;
cat header.tsv <(sort -u all_samples.vep.tab.tmp | grep -v '#' | sed 's/^chr//g' | awk -F ':'
'length($1) <= 2' | awk '{ printf "chr"$0"\n" }') > all_samples.vep.tab ;
rm all_samples.vep.tab.tmp;

postprocessing_annotation.py \
    all_samples.vep.tab \
    all_samples.vep.summary.tab \
    --assembly-name hg38 \
    --hotspots-annotation-file all_intogen_v2024.header.tsv \
    --gnomad-af-threshold 0.001 \

gzip all_samples.vep.summary.tab;
gzip all_samples.vep.tab;
```

17. Apply custom annotations to variants in the same way as it was done for the reference panels generation (CUSTOMANNOTATION). Integrates custom region annotations into the regions file that will be used as a reference. i.e. this is required for the TERT promoter to avoid TERT activating mutations benign counted as non-protein affecting.

```
mutations_custom_processing.py \
    --mutations-file all_samples.vep.summary.tab.gz \
    --custom-regions-file added_regions.tsv \
    --output-file all_samples.vep.summary.tab.custom.tsv ;
```

18. Transforms VEP annotations to MAF-like tabular format (VCF2MAF)

```
vcf2maf.py    --vcf <sample>.med.filtered.vcf \
    --sampleid <sample> \
    --annotation_file all_samples.vep.summary.tab.custom.tsv \
    --level med
```

19. Filter Variants Outside Exons (FILTEREXONS). Flags variants not within exonic regions and labels them with “not_in_exons” in the FILTER column.

```
filterbed.py \
    --sample-maf-file <sample>.med.maf.annot.tsv.gz \
    --bedfile captured_panel.exons_splice_sites.bed \
    --filtername not_in_exons
```

20. Filter variants outside consensus panel (FILTERPANEL). Flags variants not within well covered regions and labels them with “not_covered” in the FILTER column.

```
filterbed.py \
    --sample-maf-file <sample>.med.maf.annot.filtered.tsv.gz \
    --bedfile consensus.all.bed \
    --filtername not_covered
```

21. Use NanoSeq SNP Artifacts mask to flag mutations that are common germline SNPs. These are flagged with nanoseq_snp (FILTERNANOSEQSNP).

```
filterbed.py \
    --sample-maf-file <sample>.med.maf.annot.filtered.filtered.tsv.gz \
    --bedfile SNP_GRCh38.wgns.bed.gz \
    --filtername nanoseq_snp \
    --positive;
```

22. Filter regions classified as NOISE by NanoSeq (FILTERNANOSEQNOISE). Removes positions prone to technical noise as defined by NanoSeq.

```
filterbed.py \
    --sample-maf-file P19_0002_BDO_01.med.maf.annot.filtered.filtered.filtered.tsv.gz \
    --bedfile NOISE_GRCh38.wgns.bed.gz \
    --filtername nanoseq_noise \
    --positive
```

23. Merge filtered variants across samples (MERGEBATCH). Combines all variants of all the samples into a single file.

```
merge_cohort.py --output_file all_samples.cohort.tsv.gz
```

24. Apply batch-level filters (FILTERBATCH). Flags recurrent artifacts and likely germline variants at cohort level.

```
filter_cohort.py \
    --maf-df-file all_samples.cohort.tsv.gz \
    --sample-name all_samples \
    --repetitive-variant-threshold 4 \
    --somatic-vaf-boundary 0.35 \
    --n-rich-cohort-proportion 0.1
```

25. Plot unfiltered mutation statistics (PLOTMAF). Applies minimal filters before plotting: VAF ≤0.3 and depth ≥100.

```
cat > mutations_subset.conf << EOF
{
    "VAF" : "le 0.35",       "DEPTH" : "ge 100"
}
EOF

cat > requested_plots.conf << EOF
```

```
{
          "global":["per_sample",          "per_gene",          "filter_stats          SAMPLE_ID
n_rich,no_pileup_support,other_sample_SNP",           "filter_stats           canonical_SYMBOL
n_rich,no_pileup_support,other_sample_SNP",               "plot_stats               SAMPLE_ID
canonical_Consequence_broader,TYPE,MUTTYPE",            "plot_stats            canonical_SYMBOL
canonical_Consequence_broader,TYPE,MUTTYPE"]
}
EOF

plot_maf.py \
    --sample_name all_samples \
    --mut_file all_samples.cohort.filtered.tsv.gz \
    --out_maf all_samples.unfiltered \
    --json_filters mutations_subset.conf \
    --req_plots requested_plots.conf \
```

26. Write MAF file containing both germline and somatic variants (WRITEMAF). Generate a single file per sample and for each group of samples to analyze together, this step allows the independent processing of each sample. Exports all annotated and filtered variants

```
write_mafs.py \
    --maf-file all_samples.cohort.filtered.tsv.gz \
    --groups-json all_groups.json
```

27. Generate a clean mutation set (CLEANMUTATIONS). Filters applied: exclude mutations containing NM20, n_rich, cohort_n_rich_threshold, cohort_n_rich, no_pileup_support, low_mappability, not_covered or nanoseq_noise, depth ≥100× and VAF_distorted_expanded_sq = FALSE

```
cat > mutations_subset.conf << EOF
{
              "FILTER"    :    ["notcontains    NM20",    "notcontains    n_rich",    "notcontains
cohort_n_rich_threshold",   "notcontains   cohort_n_rich",   "notcontains   no_pileup_support",
"notcontains low_mappability", "notcontains not_covered", "notcontains nanoseq_noise"],"DEPTH"
: "ge 100",    "VAF_distorted_expanded_sq" : false
}
EOF

cat > output_formats.conf << EOF
{
    "header": true
}
EOF

subset_maf.py \
                --sample_name <sample> \
                --mut_file <sample>.filtered.tsv.gz \
                --out_maf <sample>.clean.mutations.tsv \
                --json_filters mutations_subset.conf \
                --req_fields output_formats.conf \
```

28. Keep only somatic mutations (SOMATICMUTATIONS). Criteria for somatic mutations: VAF ≤0.3, VAF_AM ≤0.3, vd_VAF ≤0.3, not in gnomAD (AF <0.001) and not in NanoSeq SNP mask. Samples that do not have at least 10 mutations after all this filters have been applied will not be outputted.

```
cat > mutations_subset.conf << EOF
{
        "FILTER"  :  ["notcontains  gnomAD_SNP",  "notcontains  other_sample_SNP",  "notcontains
nanoseq_snp"], "VAF" : "le 0.35",    "vd_VAF" : "le 0.35", "VAF_AM" : "le 0.35"
}
EOF

cat > output_formats.conf << EOF
{
    "header": true
}
EOF

subset_maf.py \
                --sample_name <sample> \
                --mut_file <sample>.clean.mutations.tsv \
                --out_maf <sample>.somatic.mutations.tsv \
                --json_filters mutations_subset.conf \
                --req_fields output_formats.conf \
                --min_mutations 10 \
```

29. Assess sample contamination (CONTAMINATION). Detects inter-sample contamination based on shared variant patterns. If most of the SNPs of a given sample are present in another with a VAF corresponding to a germline variant that contaminated the original sample preparations.

```
check_contamination.py --maf_path all_samples.filtered.tsv.gz \
                    --somatic_maf all_samples.somatic.mutations.tsv
```

30. Plot somatic mutation landscape (PLOTSOMATICMAF). No additional filtering of the mutations, only using the plotting functionalities.

```
cat > mutations_subset.conf << EOF
{

}
EOF

cat > requested_plots.conf << EOF
{
                        "global":["per_sample",      "per_gene",      "plot_stats      SAMPLE_ID
canonical_Consequence_broader,canonical_Protein_affecting,TYPE,MUTTYPE",           "plot_stats
canonical_SYMBOL canonical_Consequence_broader,canonical_Protein_affecting,TYPE,MUTTYPE"]
}
EOF

plot_maf.py \
    --sample_name all_samples \
    --mut_file all_samples.somatic.mutations.tsv \
    --out_maf all_samples.somatic \
    --json_filters mutations_subset.conf \
    --req_plots requested_plots.conf \
```

31. Generate needle plots for all proteins (PLOTNEEDLES). Shows mutation positions, types, and frequencies both in a needle shape but also in the form of stacked barplots with positions grouped in bins.

```
mkdir all_samples.needles ;
plot_needles.py \
                --sample_name all_samples \
                --mut_file all_samples.somatic.mutations.tsv \
                --o3d_seq_file seq_for_mut_prob.tsv \
                --outdir all_samples.needles
```

Preprocessing complete

At this stage all the inputs have been preprocessed, these are the core building steps of deepCSA.
The main files used for downstream analysis are ready and they can be summarized as:

- For each sample/group of samples for analysis deepCSA has generated:
    - A depth file (from step 13)
        - 4 columns: chromosome, position, pyrimidine-centric trinucleotide context and sequencing depth (number of reads sequenced at that specific position)
    - A file with somatic mutations and all column annotations (from step 28)
        - For each mutation information on VAF, gene, consequence, mutation type, …
- Consensus panel files created in steps 10 and optionally postprocessed at step 12.
    - These are the files with the positions that contain a minimum amount of coverage across samples and they differ between each other due to the different consequence types of the mutations in each of the positions.

In the steps further downstream each individual method may or may not require minimal formatting changes or particularities. A very usual step is the SUBSET<MUTATIONS/PANEL>.
This step takes advantage of the capabilities of tabix, and a provided BED file to subset the rows of interest to only those that belong to the desired regions. Whenever a step uses this logic we refer to their code as **TABIX-SUBSET-code**.

(i.e. MUTPROFILEALL:SUBSETMUTATIONS, DEPTHSEXONSCONS)

```
if [[ <sample>.depths.annotated.tsv.gz == *.gz ]]; then
    if [[ "1" == "1" ]]; then
        echo "mode compressed and header";
            zcat <sample>.depths.annotated.tsv.gz | tail -n +2 | sort -k1,1 -k2,2n | bgzip
--threads 12 -c  > <sample>.subset_depths.tmp.tsv.gz;
        tabix -s 1 -b 2 -e 2 <sample>.subset_depths.tmp.tsv.gz;
              cat <(zcat <sample>.depths.annotated.tsv.gz | head -n 1) <(tabix --regions
consensus.exons_splice_sites.bed -h <sample>.subset_depths.tmp.tsv.gz) | bgzip --threads 12 -c
> <sample>.subset_depths.tsv.gz;
    elif [[ "1" == "pile" ]]; then
        echo "mode pileup";
        tabix -s 1 -b 2 -e 2 <sample>.depths.annotated.tsv.gz;
       tabix --regions consensus.exons_splice_sites.bed -h  <sample>.depths.annotated.tsv.gz |
bgzip --threads 12 -c > <sample>.subset_depths.tsv.gz;
    else
        echo "mode compressed without header";
```

```
         zcat <sample>.depths.annotated.tsv.gz | sort -k1,1 -k2,2n | bgzip --threads 12 -c  >
<sample>.subset_depths.tmp.tsv.gz;
        tabix -s 1 -b 2 -e 2 <sample>.subset_depths.tmp.tsv.gz;
       tabix --regions consensus.exons_splice_sites.bed -h <sample>.subset_depths.tmp.tsv.gz |
bgzip --threads 12 -c > <sample>.subset_depths.tsv.gz;
    fi
else
    if [[ "1" == "1" ]]; then
        echo "mode uncompressed with header";
         tail -n +2 <sample>.depths.annotated.tsv.gz | sort -k1,1 -k2,2n | bgzip --threads 12
-c  > <sample>.subset_depths.tmp.tsv.gz;
        tabix -s 1 -b 2 -e 2 <sample>.subset_depths.tmp.tsv.gz;
         cat    <(head    -n    1    <sample>.depths.annotated.tsv.gz)    <(tabix    --regions
consensus.exons_splice_sites.bed -h <sample>.subset_depths.tmp.tsv.gz) | bgzip --threads 12 -c
> <sample>.subset_depths.tsv.gz;
    else
        echo "mode uncompressed without header";
             sort  -k1,1  -k2,2n  <sample>.depths.annotated.tsv.gz  |  bgzip  --threads  12  -c   >
<sample>.subset_depths.tmp.tsv.gz;
        tabix -s 1 -b 2 -e 2 <sample>.subset_depths.tmp.tsv.gz;
       tabix --regions consensus.exons_splice_sites.bed -h <sample>.subset_depths.tmp.tsv.gz |
bgzip --threads 12 -c > <sample>.subset_depths.tsv.gz;
    fi
fi
rm -f <sample>.subset_depths.tmp.tsv.gz*;
```

Some of these subsets are applied multiple times and they are followed, mainly those using somatic mutations, by an additional step that is part of the preprocessing of each individual tool that is usually called the same name as the functionality that will be used. (i.e. MUTPROFILEALL:SUBSETMUTPROFILE). These are internal steps in which the output is not stored in the deepCSA output directory, but they enable the computation of most of the analysis. Whenever a step uses this logic we refer to their code as **PYTHON-SUBSET-code**.

```
cat > mutations_subset.conf << EOF
{
    "TYPE" : "SNV"  ## here is where more filters can be added
}
EOF

cat > output_formats.conf << EOF
{
    "header": true,   "columns": ["SAMPLE_ID", "CONTEXT_MUT", "MUT_ID", "ALT_DEPTH"],
"colnames": ["SAMPLE_ID", "CONTEXT_MUT", "MUT_ID", "ALT_DEPTH"]
}
EOF

subset_maf.py \
                --sample_name <sample>.all \
                --mut_file <sample>.all.tsv.gz \
                --out_maf <sample>.all.mutations.tsv \
                --json_filters mutations_subset.conf \
                --req_fields output_formats.conf \
```

32. Takes advantage of the tabix properties to subset a TSV file only to the regions of interest for each of the corresponding sets of coordinates. It does so using the depths table provided from step 13 and the consensus panels defined along the Phase 2 (step 10).
    a. DEPTHSALLCONS
    b. DEPTHSEXONSCONS
    c. DEPTHSNONPROTCONS
    d. DEPTHSPROTCONS
    e. DEPTHSSYNONYMOUSCONS

```
TABIX-SUBSET-code x 5 each run with a different panel
```

Phase 4: Mutagenesis. Mutational profile analysis.

This phase can be computed for several different panels. With the profile_all, profile_non_prot, … options you can select which mutational profiles you want to compute. All the steps applied here will be executed with the exact same configurations, only changing the input panel and the selection of mutations accordingly.

33. Subset mutations for profile (MUTPROFILEALL:SUBSETMUTATIONS).

```
TABIX-SUBSET-code with consensus all panel
```

34. Subset mutations for mutational profile (MUTPROFILEALL:SUBSETMUTPROFILE). Generates 96-channel trinucleotide mutation matrix

```
PYTHON-SUBSET-code keeping only SNVs but not distinguishing any consequence type.
```

35. Compute mutation count matrix for the SBS96 type of matrix (MUTPROFILEALL:COMPUTEMATRIX).

```
mut_profile.py matrix \
               --sample_name <sample>.all \
               --mut_file <sample>.all.mutations.tsv \
               --out_matrix <sample>.all.matrix.tsv \
               --method unique --pseud 0 --per_sample --sigprofiler
```

36. Compute trinucleotide frequencies in the panel regions (MUTPROFILEALL:COMPUTETRINUC). It uses the trinucleotide composition and the sequencing depth of each position to compute the cumulative depth for each trinucleotide in a given genomic region.

```
mutprof_2compute_trinucleotide.py \
               --depths_file <sample>.subset_depths.tsv.gz \
               --sample_name <sample>.all \
               --pseud 1
```

37. Compute the 96-channels single base substitution (SBS) mutational profile (MUTPROFILEALL:COMPUTEPROFILE). This step generates a mutation spectra corrected for trinucleotide composition, also renormalized to the WGS trinucleotide content and plots the multiple versions that were computed.

```
mut_profile.py profile \
```

```
            --sample_name <sample>.all \
            --mutation_matrix <sample>.all.matrix.tsv \
            --trinucleotide_counts <sample>.all.trinucleotides.tsv.gz \
            --wgs --wgs_trinucleotide_counts trinuc_counts.homo_sapiens.tsv \
            --plot --wgs --sigprofiler
```

38. Concatenate All Profiles (MUTPROFILEALL:CONCATPROFILES). Combines mutational profiles from all samples and groups into a single TSV file. It computes all the pairwise cosine similarities and it generates a heatmap and clustermap with the results.

```
ls *.all.profile.tsv > mutation_profiles_list.txt
concat_profiles.py \
        --mutation-profiles-list mutation_profiles_list.txt \
        --groups-json all_groups.json
```

Phase 5: Mutagensis. Mutational Signature Analysis.

39. Convert MAF to VCF for SigProfiler (MAF2VCF) converts somatic MAF to individual VCF files per sample, but only with the mutations that passed all the filters.

```
maf2vcf.py \
    --output-dir . \
    --maf-from-deepcsa \
    --sample-name-column SAMPLE_ID
```

40. Run SigProfilerMatrixGenerator (SIGPROMATRIXGENERATOR). Generates comprehensive mutation matrices from VCF files.

```
mkdir input_mutations
cp *.vcf input_mutations/.

SigProfilerMatrixGenerator matrix_generator \
            samples \
            GRCh38 \
            input_mutations/ \
            --plot   --tsb_stat   --seqInfo  --cushion 100
```

All the steps below are executed within the subworkflow of mutational signatures (SIGNATURES.*). This can be called multiple times for exploring mutational signatures in different genomic regions.

41. Concatenate mutation count matrices after these have been renormalized to the WGS trinucleotide counts content (MATRIXCONCATWGS).

```
ls *.matrix.WGS.sigprofiler.tsv > all_files.txt;
concat_sigprot_matrices.py \
            --filename_of_matrices all_files.txt \
            --samples_json_file samples.json \
            --type_of_profile .all
```

42. Run SigProfilerAssignment using a catalog of reference signatures (SIGPROFILERASSIGNMENT). Decomposes mutation profiles into COSMIC v3.4 reference signatures, or any other set of signatures formatted similarly.

```
python     -c     "from     SigProfilerAssignment     import     Analyzer     as     Analyze;
Analyze.cosmic_fit('groups_matrix..all.sp.tsv',                    'output_groups_matrix.all',
```

```
input_type='matrix',                  context_type='96',                  genome_build='hg38',
signature_database='COSMIC_signatures.txt', exclude_signature_subgroups=[])"

mv
output_groups_matrix.all/Assignment_Solution/Activities/Decomposed_MutationType_Probabilities.
txt
output_groups_matrix.all/Assignment_Solution/Activities/Decomposed_MutationType_Probabilities.
groups_matrix.all.txt;
```

43. Compute mutation-to-signature probabilities (SIGPROBS). Assigns each mutation to its most likely contributing signature based on the trinucleotide context, the nucleotide change and the signatures of the sample in which the mutation occurred.

```
ls                                Decomposed_MutationType_Probabilities.samples_matrix.all.txt
Decomposed_MutationType_Probabilities.groups_matrix.all.txt > signature_probs_files.txt
concat_sbs_probs.py --signature-probabilities signature_probs_files.txt
```

44. De Novo Signature Extraction using HDP. All these steps are within the HDPEXTRACTION subworkflow.
    a. Prepare HDP Input (HDPEXTRACTION:PREPARE_INPUT).

```
# First, create an R script
cat <<EOF > process_data.R
data = read.table("samples_matrix..all.hdp.tsv", header = FALSE)
rownames(data) <- data[,c(1)]
colnames(data) <- data[c(1),]
data <- data[-c(1),]
data[,-c(1)] <- sapply(data[,-c(1)], as.numeric)
data <- data[,-c(1)]
write.table(data, file = "samples_matrix.all.before_round.hdp.csv")
data <- round(data)
saveRDS(data, file = "samples_matrix.all.hdp.rds")
write.table(data, file = "samples_matrix.all.hdp.csv")
EOF
# Run the R script
Rscript process_data.R

cat <<EOFF > process_metadata.R
data = read.table("samples_matrix..all.hdp.tsv", header = FALSE)
data = data[,c(1, 1)]
data <- data[-c(1),]
colnames(data) <- c("sample", "individual")
data\$group = "L"
saveRDS(data, file = "samples_matrix.all.hdp.treelayer.rds")
write.table(data, file = "samples_matrix.all.hdp.treelayer.csv")
EOFF
# Run the R script
Rscript process_metadata.R
```

    b. Run HDP Chain Sampling (15 independent chains) (HDPEXTRACTION:RUN_HDP_CHAIN_SAMPLING). Performs Bayesian signature extraction using MCMC.

```
mkdir iteration_dir
Rscript /app/HDP_sigExtraction/R/run_HDP_chainSampling.R \
    samples_matrix.all.hdp.rds \
```

```
    iteration_dir \
    samples_matrix.all.hdp.treelayer.rds \
    NA  50  10000 100  200  3  1
```

c. Process HDP Results (HDPEXTRACTION:PROCESS_HDP_RESULTS). Extracts consensus signatures from MCMC chains

```
mkdir -p output_dir/iterations/
mv hdp_chains_*.RData output_dir/iterations/.
Rscript /app/HDP_sigExtraction/R/run_HDP_processing.R \
    samples_matrix.all.hdp.rds \
    output_dir/ \
    samples_matrix.all.hdp.treelayer.rds \
    NA  50 0  0
```

d. Compare signatures to COSMIC (HDPEXTRACTION:COMPARESIGNATURES). This step matches de novo signatures to COSMIC references

```
Rscript /app/HDP_sigExtraction/R/run_HDP_comparing.R \
    output_dir/ \
    0.9     1000   0.1 \
    COSMIC_signatures.txt \
    "NA" \
    "FALSE"
cp -r output_dir samples_matrix.all.compared_output_dir
```

45. Map Mutations to Signatures (MUTS2SIGS). Use the information on the probability of each mutation type to belong to a given signature to annotate the most likely aetiology behind each mutation. This is run independently for each sample or analysis group.

```
signatures_mutations_n_sbs.py --mutations <sample>.somatic.mutations.tsv \
                  --signature-probabilities <sample>.decomposed_probabilities.tsv \
                  --output <sample>.sigs.annotated.tsv.gz
```

Phase 6: Mutagenesis & Selection. Mutation density analysis.

46. Subset mutations based on genomic coordinates.
    a. MUTDENSITYALL:SUBSETMUTATIONS
    b. MUTDENSITYPROT:SUBSETMUTATIONS
    c. MUTDENSITYNONPROT:SUBSETMUTATIONS
    d. MUTDENSITYSYNONYMOUS:SUBSETMUTATIONS

```
TABIX-SUBSET-code x 4 with the panels listed above.
```

47. Format mutations for mutation density analysis. Output columns include: SAMPLE_ID, MUT_ID, ALT_DEPTH, GENE, TYPE
    a. MUTDENSITYALL:SUBSETMUTDENSITY
    b. MUTDENSITYPROT:SUBSETMUTDENSITY
    c. MUTDENSITYNONPROT:SUBSETMUTDENSITY
    d. MUTDENSITYSYNONYMOUS:SUBSETMUTDENSITY

```
PYTHON-SUBSET-code keeping only the mutations with the corresponding consequence type in each
of the 4 calls.
```

48. Calculate Mutation Density for the different genomic regions.
    a. MUTDENSITYALL:MUTDENSITY

b. MUTDENSITYPROT:MUTDENSITY
c. MUTDENSITYNONPROT:MUTDENSITY
d. MUTDENSITYSYNONYMOUS:MUTDENSITY

```
compute_mutdensity.py \
            --maf_path <sample>.all.mutations.tsv \
            --depths_path <sample>.subset_depths.tsv.gz \
            --annot_panel_path consensus.all_with_subgenic.tsv \
            --sample_name <sample> \
            --panel_version <all|protein_affecting|non_protein_affecting|synonymous>;
```

49. Select mutations for MUTDENSITYADJUSTED computation:
    a. Subset mutations based on genomic regions for the computation of adjusted mutabilities (MUTDENSITYADJUSTED:SUBSETMUTATIONS)

```
TABIX-SUBSET-code with the consensus exons and splice sites panel.
```

    b. Format mutations for adjusted density analysis (MUTDENSITYADJUSTED:SUBSETMUTDENSITYADJUSTED)

```
PYTHON-SUBSET-code keeping only the mutations that are SNVs.
```

50. Calculate context-adjusted mutation density. This step is computationally intensive since it adjusts density for local trinucleotide composition and mutational profile. (MUTDENSITYADJUSTED:MUTDENSITYADJ)

```
mut_density.py \
                    --sample_name <sample> \
                    --depths_file <sample>.subset_depths.tsv.gz \
                    --somatic_mutations_file <sample>.mutations.tsv \
                    --mutability_file <sample>.all.profile.tsv \
                    --panel_file consensus.exons_splice_sites.tsv \
                    --trinucleotide_counts_file trinuc_counts.homo_sapiens.tsv
```

51. Reformat synonymous mutation density values per gene for omega (SYNMUTDENSITY). Mode: mutations.

```
omega_select_mutdensity.py \
            --mutdensities all_samples.synonymous.mutdensities.tsv \
            --output all_samples.gene_mutdensities.tsv \
            --mode mutations;
```

52. Reformat synonymous mutated reads density values per gene so that they can be used by omega multi mode (SYNMUTREADSRATE). Mode: mutated reads.

```
omega_select_mutdensity.py \
            --mutdensities all_samples.synonymous.mutdensities.tsv \
            --output all_samples.gene_mutdensities.tsv \
            --mode mutated_reads;
```

Phase 7: Relative mutability computations.

This measurement of the expected distribution of mutations along the consensus panel is useful for several downstream applications.

53. Subset mutations for the computation of relative mutabilities

a. MUTABILITYALL:SUBSETMUTATIONS

```
TABIX-SUBSET-code with the consensus exons and splice sites panel.
```

b. MUTABILITYALL:SUBSETMUTABILITY

```
PYTHON-SUBSET-code keeping only the mutations that are SNVs and Output: SAMPLE_ID, MUT_ID, ALT_DEPTH, SYMBOL columns.
```

54. Compute relative mutability (MUTABILITYALL:RELATIVEMUTABILITY)

```
mutprof_3compute_mutability.py \
                --sample_name <sample>.all \
                --mutation_matrix <sample>.all.mutations.tsv \
                --depths <sample>.depths.annotated.tsv.gz \
                --profile <sample>.all.profile.tsv \
                --bedfile consensus.exons_splice_sites.tsv \
                --out_mutability <sample>.all.relative_mutability_per_site.tsv \
                --adjust_local_density
```

55. Compress and index relative mutability files (MUTABILITYALL:MUTABILITY_BGZIPTABIX)

```
bgzip    --threads  1  -c    DonoridM1174003.all.relative_mutability_per_site.tsv.adjusted  > DonoridM1174003.all.adjusted.gz
tabix -b 2 -e 2 DonoridM1174003.all.adjusted.gz
```

Phase 8: Positive Selection Analysis: OncodriveFML

56. Use the consensus exons panel to generate a BED file with the regions for running OncodriveFML (ONCODRIVEFMLALL:ONCODRIVEFMLBED).

```
sh       createcustombed.sh       consensus.exons_splice_sites.tsv       oncodrivefml       > exons_splice_sites.all.annotated.bed
```

57. Subset Mutations for OncodriveFML (ONCODRIVEFMLALL:SUBSETONCODRIVEFML)

```
PYTHON-SUBSET-code keeping only the following columns CHROMOSOME, POSITION, REF, ALT and SAMPLE.
```

58. Run OncodriveFML (ONCODRIVEFMLALL:ONCODRIVEFMLSNVS)

```
[CONFIG FILE generation]
oncodrivefml -i DonoridM1174003.all.mutations.tsv \
                -e exons_splice_sites.all.annotated.bed \
                -o DonoridM1174003.all \
                --no-indels --debug \
                -c oncodrivefml_v2.mutability.conf
```

Phase 9: Positive Selection Analysis: Oncodrive3D

59. Subset Mutations for Oncodrive3D

a. ONCODRIVE3D:SUBSETMUTATIONS,

```
TABIX-SUBSET-code with the consensus exons and splice sites panel.
```

b. ONCODRIVE3D:SUBSETONCODRIVE3D

```
PYTHON-SUBSET-code keeping only the following columns: SAMPLE_ID renamed as Tumor_Sample_Barcode, SYMBOL, Consequence, Amino_acids, Protein_position, MUT_ID.
```

60. Oncodrive3D preprocessing (ONCODRIVE3D:ONCODRIVE3D_PREPROCESSING)

```
oncodrive3d_preprocessing.py \
        --maf-df-file <sample>.mutations.tsv \
        --vep-output-all all_samples.vep.tab.gz \
        --sample <sample>
```

61. Run Oncodrive3D (ONCODRIVE3D:ONCODRIVE3D_RUN). Tests for 3D clustering of missense mutations on protein structures.

```
oncodrive3D run -i <sample>.mutations.raw_vep.tsv \
                -m oncodrive3d.mutability.conf \
                -d datasets_240506 \
                -C <sample> \
                -o <sample> \
                -s 128 \
                -c 12 \
                --o3d_transcripts --use_input_symbols
```

62. Generate Oncodrive3D plots (ONCODRIVE3D:ONCODRIVE3D_PLOT). Shows 3D clustering patterns on 2D plots together with additional structural features of the protein.

```
oncodrive3D plot -o <sample> \
                  -g <sample>.3d_clustering_genes.csv \
                  -p <sample>.3d_clustering_pos.csv \
                  -i <sample>.mutations.processed.tsv \
                  -m <sample>.miss_prob.processed.json \
                  -s <sample>.seq_df.processed.tsv \
                  -d datasets_240506 \
                  -a annotations_240506 \
                  -c <sample> \
                  --output_csv \
```

Phase 10: Positive Selection Analysis: Omega. dN/dS analysis

All the following steps are executed within the OMEGA subworkflow that can be called with slightly different configurations of data. In this procedure we will use the classical estimation of dN/dS for simplicity, but we will also compute the absolute mutabilities and the site comparison measurements that allow the computation of positive selection at the level of specific sites.

63. Subset panel (OMEGA:SUBSETPANEL)

```
TABIX-SUBSET-code subsetting the captured panel with the rich annotations generated in step 6(or 7b) with the consensus exons and splice sites panel.
```

64. Subset mutations for Omega
    a. OMEGA:SUBSETMUTATIONS

```
TABIX-SUBSET-code with the consensus exons and splice sites panel.
```

    b. OMEGA:SUBSETOMEGA

```
PYTHON-SUBSET-code keeping only the following columns: CHROM, POS, REF, ALT, SAMPLE_ID.
```

    c. OMEGA:SUBSETOMEGAMULTI

```
PYTHON-SUBSET-code keeping only the following columns: CHROM, POS, REF, ALT, SAMPLE_ID and ALT_DEPTH which has been renamed to EFFECTIVE_MUTS.
```

65. Omega preprocessing step (OMEGA:PREPROCESSING). This step is essential to collect the mutation files and process them to ensure that the estimator module has all the mutabilities and number of mutations properly formatted.

```
omega preprocessing --preprocessing-mode compute_mutabilities \
    --depths-file <sample>.subset_depths.tsv.gz \
    --mutations-file <sample>.mutations.tsv \
    --input-vep-postprocessed-file consensus.exons_splice_sites_with_subgenic.tsv \
    --table-observed-muts mutations_per_sample_gene_impact_context.<sample>.tsv \
    --mutabilities-table mutability_per_sample_gene_context.<sample>.tsv \
    --synonymous-muts-table syn_muts.<sample>.tsv \
    --mutational-profile-file <sample>.all.profile.tsv \
    --single-sample <sample> \
    --absent-synonymous ignore
```

66. Prepare groups of genes for Omega (OMEGA:GROUPGENES). These groups are created based on the definition of subgenic regions (received from step 12e), or by the custom definition of new gene groups.

```
awk 'NR>1 {print $1}' syn_muts.all_samples.tsv | sort -u  > gene_list.txt

features_2group_genes.py \
    --panel-genes-file gene_list.txt \
    --add-hotspots \
    --hotspot-genes-file subgenic_names.json \
    --output-json-groups2names genes2group_out.json
```

67. Omega estimation (OMEGA:ESTIMATOR). Estimates selection coefficient ω (omega, same as dN/dS) for each gene/region and consequence type defined in the input parameters. Method: Maximum likelihood estimation (MLE).

```
mkdir groups;
mv genes2group_out.json groups/group_genes.json
cat > groups/group_impacts.json << EOF
{
    "missense": ["missense"],
    "nonsense": ["nonsense"],
    "essential_splice": ["essential_splice"],
    "truncating": ["nonsense", "essential_splice"],
    "nonsynonymous_splice": ["missense", "nonsense", "essential_splice"]
}
EOF

cat > groups/group_samples.json << EOF
{
    "<sample>" : ["<sample>"]
}
EOF

omega estimator --mutability-file mutability_per_sample_gene_context.<sample>.tsv \
   --observed-mutations-file mutations_per_sample_gene_impact_context.<sample>.tsv \
               --depths-file <sample>.subset_depths.tsv.gz \
               --vep-annotation-file consensus.exons_splice_sites_with_subgenic.tsv \
               --grouping-folder ./groups/ \
               --output-fn output_mle.<sample>.tsv \
               --option mle \
```

```
                --cores 4
```

68. Calculate absolute mutabilities using omega mutabilities command (OMEGA:ABSOLUTEMUTABILITIES). This step uses the data generated in the omega preprocessing step and converts the relative mutation probability of each site to absolute terms. This allows the assignment of a certain amount of mutations to each particular site of the gene.

```
mkdir groups;
mv genes2group_out.json groups/group_genes.json
cat > groups/group_samples.json << EOF
{
    "<sample>" : ["<sample>"]
}
EOF

omega mutabilities --mutability-file mutability_per_sample_gene_context.<sample>.tsv \
                   --depths-file <sample>.subset_depths.tsv.gz \
                   --vep-annotation-file consensus.exons_splice_sites_with_subgenic.tsv \
                   --grouping-folder ./groups/ \
                   --output-fn mutabilities_per_site.<sample>.tsv.gz \
                   --cores 4
```

69. Site comparison analysis
    Using the absolute mutabilities computed above, deepCSA compares for each site the number of mutations observed vs the expected under neutral mutagenesis. This can be done in different ways, but always follows the same input structure.

```
omega_comparison_per_site.py --mutations-file <sample>.mutations.tsv \
                     --panel-file captured_panel.subset_panel.tsv.gz \
                     --mutabilities-file mutabilities_per_site.<sample>.tsv.gz \
                     --size all \
                     --output-prefix <sample>
```

    a. OMEGA:SITECOMPARISON. Measurement of selection per site using the number of unique mutation-sample pairs as a numerator.

    b. OMEGA:SITECOMPARISONMULTI. Measurement of selection per site using the number of mutated reads (effective mutation counts) as a numerator. This metric takes into account the possible expansion of clones with mutations in a particular position of interest.

70. Omega preprocessing using globalloc[9] as the method for inference of synonymous mutation counts (OMEGA:PREPROCESSINGGLOBALLOC). This allows the definition of a stable background mutation density for all genes in all samples of a cohort. The outputs generated by this step have the same format as those from a normal omega preprocessing step and can be provided to the estimator module without any further modifications.

```
omega preprocessing --preprocessing-mode compute_mutabilities \
                    --depths-file <sample>.subset_depths.tsv.gz \
                    --mutations-file <sample>.mutations.tsv \
     --input-vep-postprocessed-file consensus.exons_splice_sites_with_subgenic.tsv \
```

```
--table-observed-muts mutations_per_sample_gene_impact_context.<sample>.global_loc.gLoc.tsv \
--mutabilities-table mutability_per_sample_gene_context.<sample>.global_loc.gLoc.tsv \
                    --synonymous-muts-table syn_muts.<sample>.global_loc.gLoc.tsv \
                    --mutational-profile-file <sample>.all.profile.tsv \
                    --single-sample <sample> \
                    --absent-synonymous infer_global_custom \
                    --mutational-profile-global-file global_mutprofile.tsv \
                    --synonymous-mutrates-file all_samples.gene_mutdensities.tsv
```

71. Omega Estimation (OMEGA:ESTIMATORGLOBALLOC)

```
Exactly the same code as above (step 67), the only difference is the source of the omega preprocessing, here is step 70 as opposed to the previous case in which it was step 65.
```

72. When using omega globalloc the working assumption is that the background mutation density per gene of the cohort reflects the background mutation density per gene of the individual samples. To ensure the biggest possible control over the generated data, this step computes several QCs comparing metrics of the observed background mutation density vs the numbers estimated using the globalloc approach. This is a critical QC and it should be checked before proceeding to use any omega globalloc values (OMEGA:EVALOMEGAGLOCESTIMATION).

```
mkdir omega_qc.plots
ls -1 syn_muts.*.tsv | grep -v 'global_loc.gLoc' > observed.txt
ls -1 *.global_loc.gLoc.tsv > estimated.txt
omega_syn_qc.py \
                --observed-syn observed.txt \
                --estimated-syn estimated.txt \
                --output-prefix omega_qc.plots/omega_qc
```

73. Calculate Absolute Mutabilities (OMEGA:ABSOLUTEMUTABILITIESGLOBALLOC)

```
Exactly the same code as above (step 68), the only difference is the source of the omega preprocessing, here is step 70 as opposed to the previous case in which it was step 65.
```

74. Site Comparison

```
Exactly the same code as above (step 69), the only difference is the source of the omega absolute mutabilities, here is step 73 as opposed to the previous case in which it was step 68.
```

a. OMEGA:SITECOMPARISONGLOBALLOC
b. OMEGA:SITECOMPARISONGLOBALLOCMULTI

Phase 11: Cohort summary plots and quality controls

75. Plot Mutation Density QC (PLOTTINGQC:PLOTMUTDENSITYQC)

```
mutation_densities_qc.py \
                --input-file all_mutdensities.tsv \
                --output-dir <sample>.plots \
                --panel consensus.exons_splice_sites.tsv \
                --group-definition all_groups.json \
                --group-name <sample>
```

76. Apply Omega QC Filters (PLOTTINGQC:APPLYOMEGAQC). This step takes the table with all omega values and the information from step 75 on which specific cases can be

affected by the inaccurate estimation of the background mutation density and flags all omega values that should not be trusted.

```
ls compiled_all_flagged_cases.*.tsv > compiled_flagging_cases.txt;
annotate_omega_failing.py \
                --omegas-file all_omegas.tsv \
                --compiled-flagged-files compiled_flagging_cases.txt \
                --output omega.flagged_annotated.tsv
```

77. PLOTTINGSUMMARY:PLOTSELECTION

```
mkdir all_samples.plots
plot_selectionsideplots.py \
                --sample_name all_samples \
                --outdir all_samples.plots
```

78. PLOTTINGSUMMARY:PLOTSATURATION

```
mkdir <sample>.plots
plot_gene_saturation.py \
                --sample_name <sample> \
                --outdir <sample>.plots \
                --domain_file domains_info.tsv \
                --exons_depths depths_per_position_exon_gene.tsv
```

79. PLOTTINGSUMMARY:PLOTINTERINDIVIDUALVARIABILITY

```
mkdir samples.variability_plots
plot_explore_variability.py \
                --mutdensities all_mutdensities.tsv \
                --panel-regions consensus.exons_splice_sites.tsv \
                --outdir samples.variability_plots \
                --samples-json samples.json \
                --all-groups-json all_groups.json \
```

Rate of errors of the technology

To obtain an approximation of the error rate of the technology, we resorted to the sequencing of cord blood DNA, as described in ref[3,9] for other DNA duplex library preparation protocols. This approach can be used because the number of mutations expected in cord blood DNA is minimal and should follow the numbers predicted by refs[1,2]. Additionally, since this has been used by others in previous studies, it allows the comparison with different techniques.

DNA duplex libraries were prepared using 150 ng of gDNA extracted with the DNeasy Blood & Tissue kit (Qiagen, cat. no. 69506) from 3 different donors of Human Cord Blood Mononuclear Cells (StemCell, Ref. 70007). Samples were fragmented as described in the main procedure and underwent end-repair, ligation to duplex sequencing adaptors, intra-strand repair and PCR amplification. The PCR product was captured for 16–20 h with a 192 gene panel, with commonly mutated cancer genes, and sequenced in a NovaSeq X.

FASTQ files were used to run deepUMIcaller as described in the main protocol and obtained the mutation calls for each of the three samples that were further processed with deepCSA to compute an estimation of the mutation density of those samples in the targeted regions. To compare with other experiments we proceeded to rescale the mutation density to the WGS scale.

Extended Data Figure 1 shows the rate of somatic variants detected across three cord blood samples using the DNA library preparation described in the DeepClone protocol and that reported by targeted Nanoseq[3]. The error rate calculated for the DNA library preparation in DeepClone is in the order of $2.5x10^{-8}$ errors per duplex nucleotide. This is very similar to the value reported by other DNA duplex sequencing protocols (Supp. Table 1).

Data QCs to analyze prior to studies of mutagenesis and selection

Here we summarize a recommended set of QCs that must be taken into consideration before proceeding with the analysis of the results of methods that compute measurements of mutagenesis and selection.

Metrics to evaluate

In this section we show some examples of exploratory plots that can help the users decide when they can or cannot use some of these specific variables, and the influence these may have in downstream analyses.

Sequencing depth

The depth of sequencing defines the resolution of the duplex sequencing technology. The higher the depth, the bigger the number of mutations that will be detected. It is essential to take this into account when making any comparison between samples or genes, since biases in the sequencing depth will confound any downstream analysis. Explore the results in the depthssummary directory part of the deepCSA output for more details on your cohort (See Extended Data Fig. 2 and 3a,b). The specific thresholds on minimal depth for mutations and for the consensus panel can be cohort specific, but none of the two should be smaller than 100.

Variant allele frequency

The variant allele frequency (VAF) is computed as the ratio between the count of mutated reads over the count of all reads sampled at a given position. In a scenario where the number of reads sampled at a given position is small, the resulting VAF can largely overestimate the actual frequency of that mutation in the tissue (Extended Data Fig. 3a,b).

In the two plots below you can observe the hyperbolic curves of the relationship between VAF and DEPTH with the count of mutated reads is 1, 2, 3 and so on, respectively. The amount of

mutations inflated and in the highest side of the hyperbolic curves can be addressed by updating the VAF measurement to that obtained from the information coming from all the unique molecules (BAM_unique) or alternatively the user can set a minimum depth threshold that can be used for downstream comparisons.

Thus, comparisons of the VAF computed for independent mutations across samples or genes is discouraged except in situations in which the depth is very homogenous along the target regions and samples sequenced. In the case of genes with very heterogeneous depth (e.g., with exons that are difficult to capture or align) the VAF of mutations should not be summed. Across samples, differences in depth are inevitable due to inter-experimental variation. However, this problem can be potentially solved by downsampling the sequencing reads to obtain the same mean average depth across samples.

Mutation density

The same QCs and explorations can be applied to mutation density estimations. These may also be obtained from small numbers of mutations and the accumulated depth at a given gene could be ambiguously inflating or deflating the mutation density. See Extended Data Fig. 3c where using the mutation density per gene-sample directly will be correct for a big part of the genes, which have big cumulative depths, but for those with little cumulative depth the measurements are likely either an over (high values) or an under estimation (0s).

This exploration is particularly critical for the synonymous mutation density of these small genomic elements since this is used for inferring the background mutation density. For very sparse values of protein affecting mutations this could also lead to overestimation of some values of positive selection per sample (Extended Data Fig. 3c).

Cohort-level gene analysis

At the level of cohort, one of the most important assumptions to study selection at the gene level is that the observed synonymous mutation density of a gene represents its background mutation density. This value is used to compute the dN/dS ratios (see the description of omega and other metrics of positive selection in ref[9]). An incorrect estimation of the background mutation density could lead to false positive or false negative results.

To assess if these values can be trusted, we state that the background mutation density of every gene should be similar and the distribution should approach a normal distribution. We automatically check this assumption (see deepCSA step 75 above) by taking the synonymous mutation density of every gene at the cohort level (only SNVs) and plotting the distribution of values. We use the log10 converted mutation densities to compute a Z-score and classify as negative or positive outliers those genes whose synonymous mutation density does not fall within a value smaller than 2 (Extended Data Fig. 5a).

After this outlier definition, any values of omega computed for those outlier genes should not be trusted. We postprocess the table with all the omega results and the QCs of mutation densities

to generate an omega output with additional information on cases that should not be trusted. This is available at applyomegaqc directory in the deepCSA output.

In addition to synonymous mutation density, we also check the non-protein affecting mutation densities, which include synonymous mutations as well as mutations detected in the non-coding regions surrounding exons. This metric includes a larger number of mutations and thus it provides a more robust estimate to assess genes identified as outliers based on synonymous mutations only.

All these outputs can be found in the plotmutdensityqc directory in the deepCSA output directory.

Sample-level analysis

Following the same principles and methods as for the gene-level mutation densities, we aim to assess whether the background (non-protein affecting) mutation density of any sample substantially differs from the rest of the samples in the cohort. As opposed to the case of genes, here we have to consider that each sample may be of a different nature so the amount of disagreement here might be bigger and have underlying biological reasons (Extended Data Fig. 5b). The outputs for the sample-level analysis can also be found at the plotmutdensityqc directory in the deepCSA output directory.

Additionally to these background mutagenesis comparisons, we also provide a comparison of the mutational profiles of the different samples and an assessment of interindividual variability.
The comparison of mutational profiles across samples, via the computation of cosine similarities between all possible sample pairs is available at the concatprofiles output directory. This enables the quick visual representation of all the mutational profiles, and can help understand the differences in the mutation density or mutational signatures measurements (Extended Data Fig. 4a,b).

The assessment of interindividual variability also takes into account the distinction between different genes and can help provide a measurement of the sparsity of the data, or identify individual/group specific differences that are very obvious.

When computing and checking the listed QCs it is important to keep asking questions regarding the possible natural or artificial origin of the results. Non-biological explanations for unexpectedly high/low mutation densities should be assessed with mutational signatures analysis and could be due to the presence of processing artifacts, or numerical artifacts product of undersampling the targeted regions.

As part of the mutational signatures analysis, the user can extract or map signatures that resemble those from COSMIC. However, when working with very small numbers of mutations (<100) it is preferable to not overinterpret the results and resort to bigger or richer cohorts to compile all this information.

Sample-gene-level analysis

These are the trickiest and more likely to be unstable metrics since the cumulative coverage achieved at gene-sample level is many times not sufficient to sample multiple mutations of every consequence type. Truncating and synonymous sites are the least abundant so they are usually the most sparse measurements.

The values of background mutation density will be too sparse causing the per gene-sample estimation of this value to be too unstable. This requires other solutions to estimate the background mutagenesis in order to accurately model positive selection. A potential solution to this problem relies on the assumption that the relative distribution of synonymous mutations between genes is similar across samples, thus we can use the cohort level background mutation density to inform the distribution of synonymous mutations in each sample. This approach was used in Calvet & Blanco Martinez-Illescas et al.[9] enabling the computation of positive selection per sample. Note that this is only possible if the cohort-level mutation density is stable enough for that gene.

Despite knowing that this assumption is safe in multiple cohorts, deepCSA generates minimal QCs to assess the deviation of omega globalloc on the number of synonymous mutations that is used. The main QC is to check the correlation between the number of synonymous mutations observed in a sample/gene and the number inferred by omega globalloc (Extended Data Fig. 6).

The second focus of the sparsity is on the number of protein affecting, particularly truncating mutations, but it can also happen with missense mutations in samples that have not been sequenced very deep (<1,000x). Although there is no direct solution to increase these numbers other than sequencing deeper, the user can quantify the frequency of samples with 0, 1, 2 or these small numbers of mutations and only cases where most of the samples are mutated would result in trustworthy sample-gene-level metrics.

Other consideration when analyzing DNA duplex sequencing data

The tools to analyze mutagenesis and selection rely on a series of assumptions. Heavy deviations from these assumptions may result in inadvertent biases introduced in these analyses. It is important, therefore, to be aware of them. With respect to the mutational profile and relative mutational probabilities, every mutation is counted only once in a sample (independently of the number of reads supporting it). This is based on the conservative assumption that mutations contributed by more than one DNA fragment are, in reality, the result of a clonal expansion. It is still possible that some of these mutations are the result of convergent evolution, rather than clonal expansion. The calculations of omega based on this assumption are, therefore, in any case, an underestimation. The calculation of the background mutation density, used for the estimation of positive selection using omega, assumes that the rate of mutations along the entire gene is uniform. If this is not the case, the results of positive selection metrics for subgenic regions could be misleading. It also assumes that the relative mutation density for the gene across samples is uniform.

Since the protocol is targeted (i.e., only a small fraction of the genome is sequenced), all analyses assume that the mutational processes observed in the targeted region represent those active across the entire genome (see main manuscript). The positive selection of indels is estimated using an ad hoc method that does not rely on the comparison to the expectation, as this is very hard to estimate for indels[9].

Project design

In this section, we discuss several limitations or elements of the experiment that need to be considered when planning a project using DNA duplex sequencing.

External constraints

As in all studies, the source of the data is critical. In particular, for duplex DNA sequencing, the main factors defining the samples of a cohort are: the number of samples(/donors/experiments) that can be collected, how much DNA can be obtained from each of these samples and which is the purity of the extracted DNA.

For duplex sequencing there is no minimal nor maximum theoretical amount of DNA, but this will directly affect the sequencing depth that will be obtained per sample. To ensure that the average depth per sample is at least 3,000x, it would imply that the DNA duplex library should start with at least 300 ng.

The purity here does not refer to tumor-normal contamination but rather to a quantification of how overrepresented is the DNA of the cells that are the target of the study. When planning the sampling, the amount of immune infiltration and the different cell types present in the sample should be taken into account.

Technical considerations

The diversity of clones that can be detected with duplex sequencing will depend greatly on the identity of the tissue that is being sampled, and the type of sampling. If the available sample is not very diverse, and few clones can be represented in the extracted DNA, then the duplex depth required per sample would be small since it will easily saturate the detection of new mutations. Opposingly, if the sample is very diverse, increasing the sequencing depth will result in detecting more new mutations of the smaller and underrepresented clones.

The cost of duplex sequencing lies mostly on the side of sequencing the samples to ensure that enough duplicates are available to build duplex consensus reads. Thus, for most of the questions it is critical to design the scientific questions of interest carefully to keep the cost under control. For studies aimed at discovering new genes under selection it is challenging to make a decision on which ones to include in the design of a hybridization capture panel, and resorting to a broad panel of genes or, if the cost allows an exome panel, could be the solution. In those cases, sequencing one or very few samples will likely not be very informative unless

the selection signal of some genes is very strong, so taking this into account we recommend that at least 10 samples are sequenced in order to detect lower level signals.

When the goal is to identify potential exposures or clinical variables that create differences in mutagenesis or selection between groups, the decision on the regions to sequence and the target sequencing depth can be much more informed on the previous knowledge of the tissue or samples that will be processed. For mutagenesis questions, it is critical to have a good representation of all possible trinucleotide changes, and if possible, have them in neutral positions of the genome to avoid any potential influence of positive or negative selection. For questions focussed on positive selection, it will be important to have a good representation of the positions that can be mutated and are subject to selection but it is also critical to include enough synonymous sites so that the background mutation density can be properly estimated.

We provide a script in our deepCSA repository (https://github.com/bbglab/deepCSA/blob/dev/assets/useful_scripts/assess_panel.py) so that users can provide some details on the experiment design, including mutation rate of the tissue, number of samples, target sequencing depth and a file with the regions that they are planning to include in their panel, and they would obtain an estimated number of mutations detected for each of the genes at the level of cohort and also per sample. These numbers can serve as an indication of how the panel should be changed to ensure that the genes have enough synonymous mutations.

Note that for studies where the samples might acquire very few mutations, the error rate of the selected duplex technology will be very important to ensure that any detected signal is driven by the real mutations and not errors induced by the duplex sequencing protocol. It is known that the mutation rate of the adult tissues is likely much higher than the error rate of the technology, at least it was shown to be the case for the normal bladder urothelium[9].

Estimation of optimal sequencing output

Here we explain how to estimate the optimal sequencing output for a duplex library, that is, the optimal number of raw reads that need to be allocated to each sample to enable duplex sequencing by the production of duplex reads. The sequencing output is directly proportional to the input DNA: the more DNA in the library, the more sequencing reads will be needed. However, the calculation is not direct, as multiple factors determine the efficiency of conversion from original DNA molecules to duplex reads. Here we present two approaches to perform this calculation. The first one is based on the calculation of unique DNA fragments estimated by the qPCR and relies on theoretical approximations of the duplex library preparation. Therefore, it does not require any prior sequencing information. The second one is specific to each duplex library preparation sequencing performance, and it gains importance when assessing how optimally sequenced the duplex library was.

Method 1: From unique DNA fragments estimation

This approach only requires knowing the unique DNA fragments estimated by the qPCR and the panel size. For the rest of the metrics described below, some theoretical approximations can be used assuming optimal library performance. However, if metrics from previous duplex library preparations with the same experimental design are available, these metrics can be used for a more accurate extrapolation of duplex library performance. It is advised to generate compiled versions of all these metrics and plots by running the scripts provided in the corresponding protocol section.

First, to calculate the amount of Unique DNA fragments, we need to know the concentration in pg/µL estimated by qPCR. Assuming the duplex library is in an elution volume of 20 µL at this step of the protocol, the molarity in fmol of the Unique DNA fragments of each DNA duplex library can be calculated as follows:

$$Unique\ DNA\ fragments\ (fmol) = Quantity\ (\frac{pg}{\mu L}) \cdot DF \cdot \frac{1\ mol}{660 \cdot 10^{12}\ pg \cdot Size\ (bp)} \cdot \frac{452\ bp}{Size\ (bp)} \cdot \frac{10^{15}\ fmol}{1\ mol} \cdot 20\ \mu L$$

Where DF is the Dilution Factor used to prepare the qPCR reaction in order to be within the quantitative range of the KAPA Library Quantification kit, Size is the duplex library size at this step of the protocol and 452 bp is the size of the standards used in this kit. The Unique DNA fragments (fmol) is converted to Unique DNA fragments (molecules) by multiplying fmol by the Avogadro's number.

The Unique DNA fragments (molecules) is then used to calculate "All Unique Molecules (AllUniMol)", which represents the total amount of unique molecules that will be sequenced, accounting for the fact that most molecules will be on target, but not all. It can be computed with this formula:

$$AllUniMol = Unique\ DNA\ fragments \cdot Panel/Genome \cdot CorrectionDry/Wet \cdot \frac{1}{OnTarget}$$

Where the metrics used are:

- Panel/Genome ratio: Extended Panel Size in bp / Genome size in bp
- Extended Panel Size: panel size in bp + number of exons in the panel · 250 bp of captured flanking regions · 2 (to account for both DNA fragments' ends)
- Correction factor DryLab/WetLab (CorrectionDry/Wet): it indicates how much quantification error there is in estimating the unique DNA fragments with the qPCR, compared to the observed unique molecules sequenced. It depends on the qPCR thermocycler instrument and laboratory. For our set-up, this value is around 2.5. This value was obtained after comparing the qPCR estimations of unique molecules vs the detection of unique molecules when sequencing over several experiments.
- On-target ratio of Unique molecules (OnTarget): proportion of unique molecules which are within the target panel, out of the total unique molecules sequenced. Assuming an optimal capture, this value should be around 0.8, however this will vary depending on the capture's efficiency.

- Desired Copies (DesCop): minimal PCR amplification of the unique DNA fragments needed to be able to call a duplex family. This value theoretically would be 10, meaning we need 5 copies of each strand of DNA to be able to call a duplex consensus. However, the reality is that in the process of the DNA duplex library preparation, the amplification is not uniform and one strand of DNA might get more copies than the other. To ensure that for most unique molecules, both strands have at least 5 copies, we account for an excess, which is typically set at 17-20.

- Amplification bias On/Off (AmpBiasOn/Off): how biased is the PCR amplification post-capture to amplifying DNA fragments which are on-target, meaning within the target panel. It can be determined with the integrated duplex metrics, from previous experiments using the same target panel and the same number of captures. If no previous information is available, this value could be close to 4 if the capture was successful.

Taking this into account, we can estimate the optimal sequencing output (Opt Seq) in number of reads or Gigabases, with the following formulas:

$$Opt\ Seq\ (reads) = AllUniMol \cdot OnTarget \cdot DesCop + (AllUniMol \cdot (1 - OnTarget) \cdot \frac{DesCop}{AmpBiasOn/Off})$$

To convert this value to Gigabases for paired-end 150 Illumina sequencing,
$Opt\ Seq\ (Gbs) = Opt\ Seq\ (reads) \cdot 150\ bp \cdot 2 \cdot 10^{-9}$.

If the quantification of Unique DNA fragments with the qPCR is not available, an estimation of sequencing output can still be calculated. In this case, a recovery of 5-6% of Unique DNA fragments can be assumed from the input DNA.

Method 2: From sequencing information of the same sample

This approach requires a sample to be sequenced for computing which would be the optimal amount of sequencing required. The formula behind the computation is also using the total unique molecules available, the proportion of unique molecules on target, and the difference in amplification between the molecules on target and those off target. However, in this case all these values are obtained directly from the sequencing data and the resulting value is specific for each sample.

These are all the metrics that can be obtained from deepUMIcaller that are then used by the WetDry metrics compilation scripts to compute the optimal sequencing. See: https://github.com/bbglab/wetdry-metrics/blob/a4eb80c99a319dfba64277bf1402058df1669175/scripts/BuildDryLabMetricsTable.py#L182-L220

- Total number of on-target of unique molecules (OnTargetMolecs): the number of molecules counted in the family metrics step when running with the on target data (see deepUMIcaller extended protocol step 11d)

- Total number of off-target unique molecules (OffTargetMolecs): computed as the subtraction between the total number of molecules counted in the family metrics step when running with all the molecules (see deepUMIcaller extended protocol step 11b) and the number of OnTargetMolecs.

- Amplification bias On/Off: this has been explained above and it captures the difference in amplification of the molecules that are on target vs those that are off target. This is computed as:

$$AmpBiasOn/Off = \frac{(Total\ on\ target\ reads\ /\ OnTargetMolecs)}{(Total\ off\ target\ reads\ /\ OffTargetMolecs)}$$

As it can be seen in the formula, we aim to quantify how more (or less) likely we are to sequence a read belonging to an on target molecule compared to a read belonging to an off target molecule. We have observed that the number of reads used for building SSCs for on target reads tends to be bigger than the number of reads used for building SSCs for off-target molecules (Extended Data Fig. 7).

Once we have all these metrics computed, the optimal number of reads required can be computed as:

$$Opt\ Seq\ (reads) = OnTargetMolecs \cdot DesCop + (OffTargetMolecs \cdot \frac{DesCop}{On/Off})$$

To convert this value to Gigabases for paired-end 150 Illumina sequencing,
$Opt\ Seq\ (Gbs) = Opt\ Seq\ (reads) \cdot 150\ bp \cdot 2 \cdot 10^{-9}$.

Metrics of mutagenesis and selection

DeepCSA computes several metrics of mutagenesis and selection that can be used as variables to regress against donors' lifetime exposures, and thus study their roles as mutagens and/or promoters.

The most useful metrics of mutagenesis computed by deepCSA concern the relative and absolute activity of different mutational processes. These can reveal which donors have been exposed to exogenous mutagens, such as tobacco smoking, alcohol or certain types of chemotherapy. The activity of a signature of unknown origin across samples can be regressed against known exposures as a means to understand its etiology.

Several metrics of the active forces of selection in a tissue are also included in deepCSA. For example, the comparison of the protein affecting (e.g., missense, nonsense, splice-affecting) mutation density of different genes and their non-protein affecting (e.g., synonymous, intronic) mutation density provides a proxy measurement of the positive selection acting on them. The stronger positive selection on the mutations of a gene in a sample will be apparent by a higher

imbalance in the protein-to-non-protein affecting mutation densities. A more direct measurement of the positive selection on the mutations of different genes is their dN/dS magnitude, which compares the density of observed mutations in a gene with that expected under neutrality. Other metrics of positive selection of the mutations in a gene included in deepCSA (e.g., the functional mutation bias calculated by OncodriveFML[16]) also provide measurements of the clonal landscape of a tissue sample. Regressing these metrics of selection against donors' known exposures can point at which of them act as promoters of pre-mutated clones.

More details about different metrics of mutagenesis and selection can be found in ref[9] and in the deepCSA GitHub repository.

Association analyses between clonal landscape metrics and donors' exposures

In this section, we describe the methodology and discuss important considerations for conducting association analyses between clonal structure metrics and lifetime external exposures or relevant clinical data (hereinafter "exposures"). This type of analysis should only be followed after proper assessment and exploration of the clonal structure metrics, both in terms of confidence (*i.e.* reliable estimates were calculated, see Supplementary Note on Data QCs to analyze prior to studies of mutagenesis and selection) and variability (*i.e.* interindividual variability observed across samples for specific exposures).

Feature selection

Ultradeep duplex sequencing of normal tissues is often not scalable to more than a hundred samples. For this reason, careful selection of genes, metrics and exposures for which to test a potential association is a critical step: due to the limited sample size, multiple testing correction can result in insufficient statistical power to uncover true associations.

Genes and metrics

Somatic mutagenesis and selection are the two fundamental evolutionary forces that shape the clonal structure of a tissue. There are three quantitative surrogates of the clonal landscape that we can derive from the ultradeep duplex sequencing data, which we recommend examine initially for association analysis with exposures:

- Non-protein affecting mutation density: in principle not subject to selection, constituting a good proxy of the contribution of neutral mutagenesis to the clonal structure of the tissue.
- Protein-affecting mutation density: enriched in variants more likely to be under selection, constituting a good proxy of the contribution of selective forces to the clonal structure of the tissue, though it is likely also affected by neutral mutagenesis.

- dN/dS (omega method calculation): most accurate measurement of selection (both negative and positive) as it estimates the excess of non-synonymous mutations relative to the expectation under neutrality.

As a starting point, one can focus the analysis on genes that appear clearly selected at the cohort level by dN/dS measurement, especially when using large target panels. Next, if the user has defined groups in their analysis based on relevant clinical variables or exposures (*e.g.* males *vs* females, ever smoker *vs* never smokers, etc.), it is recommended to examine the difference in the metric estimates per group. This does not guarantee a true association (or the absence thereof) between the metric and the exposure, but it serves as a useful starting point to navigate the potential associations to be tested. Nevertheless, do not discard examining other genes and exposures you think may be interesting in posterior iterations.

Following on from previous sections, estimating the magnitude of positive selection per gene per sample may not always be possible. Also, the calculation of mutation density, though feasible, can be biased by a sparsity of mutations that may not be easily accounted for (Extended Data Fig. 3c). Given that a minimum number of samples representing each exposure group is required to conduct a meaningful association analysis, we recommend performing this quality check when deciding which combinations of genes, metrics, and exposures to analyze.

Exposures

Exposure selection should be prioritized based on aspects such as annotation reliability, balance across the cohort, and correlation with other exposures. The correlation between continuous variables can be assessed using Pearson correlation, while correlation involving at least one categorical variable can be evaluated via logistic regression using McFadden's pseudo-R2. It is recommended to avoid including highly correlated variables when they encapsulate an indiscernible effect (*e.g.* having a cancer history and undergoing cancer treatment).

Linear models following a two-step strategy

We recommend following a two-step strategy of univariate followed by a more stringent multivariate analysis to test the associations between clonal structure metrics and exposures. As mentioned above, the generally small sample size poses a challenge in avoiding false negative associations due to the limited statistical power of the cohort, so the number of tests should be carefully selected. For the same reason, spurious findings can occur, so every association should be thoroughly examined and assessed for robustness. Additionally, it is possible to leverage linear mixed-effects models (LMEM) to include multiple samples from the same individual and increase statistical power. LMEMs are composed of both fixed and random effects and are suitable for cases in which there is not complete independence between data points. The fixed effects in these models are the exposures whose associations are to be evaluated, while the random effect is the individual to which the sample belongs (random intercept). The variability produced by the random effects is incorporated into the model, meaning it does not confound the fixed effects.

We first build univariate linear models to test the independent associations between the surrogates of the clonal structure and the exposures, fitting one model per metric, gene and exposure combination. We recommend correcting for multiple testing and deeming significant all associations with a corrected p-value of 0.2 or below. This 20% false discovery rate (FDR) cutoff allows for not missing real associations for which the cohort might be slightly underpowered. Because the surrogates of the clonal structure are intertwined, therefore not statistically independent from each other, FDR correction can be carried out for each surrogate separately. As some exposures might be confounded with one another, all associations reaching significance should be verified in a more stringent multivariate analysis. We recommend considering as confounders any correlated exposures, as well as any other significant exposures from the univariate analysis. Performing this two-step strategy, in which univariate tests are initially conducted instead of a direct multivariate analysis with all the exposures, increases the chance of uncovering real associations and mitigates the multiple testing burden. To assert significance in the multivariate analysis, we again recommend a 20% FDR cutoff.

Note that when including age as a covariate it is recommended to set the intercept of the model to zero, as we do not expect neither mutagenesis nor selection to shape the tissue (or otherwise do it minimally) before birth.

Implementation in bbgregressions

The aforementioned rationale is implemented in bbgregressions, a software package designed to manage and execute this type of customized statistical regression analysis in a contained manner. It provides a modular, end-to-end pipeline spanning configuration setup and data preparation to model execution and results visualization. DeepCSA incorporates bbgregressions as a final optional step of the analysis, in two versions:

- Default: a first-pass exploration of potential associations using the set of genes with higher positive selection across the cohort level that also pass the minimum quality filters per sample.
- Customized: after the user explores the data and conducts association analyses with bbgregressions outside of deepCSA, it is possible to provide the exact configuration employed to the specific deepCSA run, ensuring that all downstream analysis remains integrated.

Documentation on how to install and use bbgregressions outside of deepCSA, together with an extensive explanation of the configuration, commands, required inputs, and outputs is available in the GitHub repository at https://github.com/bbglab/bbgregressions.

Interpretation of results and robustness

Following the two-step strategy, we obtain one multivariate model per gene and metric combination. This model is composed of several elements (output in separated files by bbgregressions):

- Coefficients of all the exposures included in the refined multivariate analysis. As these are linear regressions, the coefficients can be interpreted as the size of the effects each exposure exerts over the clonal selection metric magnitude (per gene) over the baseline effect (*e.g.* size of the effect difference between never smoked, the baseline, and ever smoked).
- Coefficients' confidence intervals.
- Coefficients' p-values and q-values, if multiple testing correction is applied.
- Estimated model intercept.

This information can be visualized as a coefficient plot, which is a default output of bbgregressions.

As we insist above, this type of analysis can be hampered by a small cohort size. Sometimes, big positive or negative effects are observed with large confidence intervals that overlap zero. This might be indicative of lack of statistical power rather than inexistant effect. Likewise, the robustness of any significant effect should be addressed to avoid artefactual associations. Some recommendations:

- Incorporate technical variables as covariates, such as the sequencing depth, the sequencing machine (if there is suspicion of differential rate of artefactual mutations across machines used in the project), the DIN, the postmortem days before sample collection (if using autopsy samples), and the number of days spanning from sample collection to DNA extraction, among others.
- As unbalanced groups can bias associations towards the bigger group, the analysis can be repeated in a smaller cohort with a balanced number of samples per group. This should be performed in at least 10 different randomly subsampled cohorts to account for sampling variability.
- Re-do the analysis discarding samples whose mutation density is below the error limit of the technology, as these samples could carry artifact mutations that lead to incorrect selection estimates.

References

1. Osorio, F. G. *et al.* Somatic Mutations Reveal Lineage Relationships and Age-Related Mutagenesis in Human Hematopoiesis. *Cell Rep.* **25**, 2308-2316.e4 (2018).
2. Machado, H. E. *et al.* Diverse mutational landscapes in human lymphocytes. *Nature* **608**, 724–732 (2022).
3. Lawson, A. R. J. *et al.* Somatic mutation and selection at population scale. *Nature* **647**, 411–420 (2025).
4. Salk, J. J., Schmitt, M. W. & Loeb, L. A. Enhancing the accuracy of next-generation sequencing for detecting rare and subclonal mutations. *Nat. Rev. Genet.* **19**, 269–285 (2018).
5. Kennedy, S. R. *et al.* Detecting ultralow-frequency mutations by Duplex Sequencing. *Nat. Protoc.* **9**, 2586–2606 (2014).
6. Schmitt, M. W. *et al.* Detection of ultra-rare mutations by next-generation sequencing. *Proc. Natl. Acad. Sci.* **109**, 14508–14513 (2012).
7. Abascal, F. *et al.* Somatic mutation landscapes at single-molecule resolution. *Nature* **593**, 405–410 (2021).
8. Nandi, S. P. *et al.* A Universal Duplex Sequencing Approach for Accurate Detection of Somatic Mutations. 2025.09.14.676103 Preprint at https://doi.org/10.1101/2025.09.14.676103 (2025).
9. Calvet, F. *et al.* Sex and smoking bias in the selection of somatic mutations in human bladder. *Nature* **647**, 436–444 (2025).
10. Krimmel, J. D. *et al.* Ultra-deep sequencing detects ovarian cancer cells in peritoneal fluid and reveals somatic TP53 mutations in noncancerous tissues. *Proc. Natl. Acad. Sci. U. S. A.* **113**, 6005–6010 (2016).

11. Salk, J. J. *et al.* Ultra-Sensitive TP53 Sequencing for Cancer Detection Reveals Progressive Clonal Selection in Normal Tissue over a Century of Human Lifespan. *Cell Rep.* **28**, 132-144.e3 (2019).

12. Matas, J. *et al.* Colorectal Cancer Is Associated with the Presence of Cancer Driver Mutations in Normal Colon. *Cancer Res.* **82**, 1492–1502 (2022).

13. Neville, M. D. C. *et al.* Sperm sequencing reveals extensive positive selection in the male germline. *Nature* **647**, 421–428 (2025).

14. Cheng, Y. *et al.* Improved Mutation Detection in Duplex Sequencing Data with Sample-Specific Error Profiles. *BioRxiv Prepr. Serv. Biol.* 2025.07.13.664565 (2025) doi:10.1101/2025.07.13.664565.

15. Yokoyama, A. *et al.* Somatic mosaicism in the buccal mucosa reflects lifestyle and germline risk factors for esophageal squamous cell carcinoma. *Sci. Transl. Med.* **17**, eadq6740 (2025).

16. Mularoni, L., Sabarinathan, R., Deu-Pons, J., Gonzalez-Perez, A. & López-Bigas, N. OncodriveFML: a general framework to identify coding and non-coding regions with cancer driver mutations. *Genome Biol.* **17**, 128 (2016).