ALITHEA GENOMICS

MERCURIUS™

DRUG-seq and BRB-seq Data Analysis Guide

PN 10841, 10851, 11041, 11051

June 2026

Table of Contents

Protocol Brief Overview

This guide describes the data pre-processing workflow for MERCURIUS™ DRUG-seq and BRB-seq libraries.

MERCURIUS™ DRUG-seq enables the preparation of Illumina-compatible 3′ RNA-sequencing libraries from up to 1,536 cell lysate samples (2,000–50,000 mammalian cells per well) in a time- and cost-efficient manner. The kit includes a mild cell lysis buffer that allows crude cell lysates to be used directly in the reverse transcription (RT) reaction, eliminating the need for labor-intensive RNA purification. MERCURIUS™ BRB-seq follows the same workflow but starts from purified RNA rather than cell lysates.

Each kit contains barcoded MERCURIUS™ Oligo-dT primers that tag poly(A)+ RNA molecules with sample-specific barcodes during first-strand cDNA synthesis. This enables cDNA generated from all samples within an experimental group to be pooled into a single tube, greatly simplifying downstream library preparation and reducing processing costs.

The workflow is highly flexible and supports pooling of any number of samples up to the capacity of the supplied plate format (96, 384, or 1,536 wells).

Library indexing is performed using a Unique Dual Index (UDI) strategy, minimizing the risk of index hopping and barcode misassignment during next-generation sequencing. Each kit contains four UDI adapter pairs, allowing preparation of up to four independently indexed libraries. Libraries generated with different UDI adapters can subsequently be pooled and sequenced together on the same flow cell.

For simplicity, the remainder of this document refers to MERCURIUS™ DRUG-seq. All described data processing steps are fully applicable to MERCURIUS™ BRB-seq libraries.

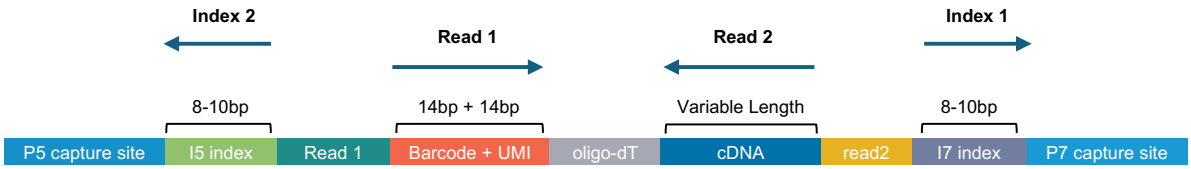

Figure 1: Schematic representation of MERCURIUS™ DRUG-seq library structure. Read 1 includes the barcode and UMI sequences. Read 2 contains the cDNA information. Index 1 and Index 2 are 8 (regular) or 10 bp (XLEAP).

Related Products

Kit name	Kit PN
MERCURIUS™DRUG-seq Library Preparation 96 kit	10841
MERCURIUS™ DRUG-seq Library Preparation 384 kit	10851
MERCURIUS™ DRUG-seq Library Preparation 384 (4x96) kit	11041
MERCURIUS™ DRUG-seq Library Preparation 1536 (4x384) kit	11051
MERCURIUS™ BRB-seq Library Preparation 96 kit	10813
MERCURIUS™ BRB-seq Library Preparation 384 kit	10814
MERCURIUS™ BRB-seq Library Preparation 384 (4x 96) kit	11013
MERCURIUS™ BRB-seq Library Preparation 1536 (4x 384) kit	11014
MERCURIUS™ HS BRB-seq Library Preparation 96 kit	10881
MERCURIUS™ HS BRB-seq Library Preparation 384 kit	10891
MERCURIUS™ HS BRB-seq Library Preparation 384 (4x96) kit	11881
MERCURIUS™ HS BRB-seq Library Preparation 1536 (4x384) kit	11891
MERCURIUS™ DRUG-seq Library Preparation Kit for 1536 Well-Plates	11091
MERCURIUS™ Spheroid DRUG-seq Library Preparation 96 kit	10870
MERCURIUS™ Spheroid DRUG-seq Library Preparation 384 kit	10875
MERCURIUS™ Spheroid DRUG-seq Library Preparation 384 (4x 96) kit	11670
MERCURIUS™ Spheroid DRUG-seq Library Preparation 1536 (4x 384) kit	11675

Suggested Hardware – Pre-Processing

Resource requirements depend on several factors such as the total number of reads and the size of the reference genome. The figures below are indicative values for the full user guide sequence, although resource requirements are dominated by the STARsolo alignment and genome creation steps.

Small dataset (<1.5 billion reads)

- **CPU / Threads**: ≥8 CPU
- **RAM:** ≥50 Gb RAM
- **Storage**: ≥400 Gb

Large dataset (~7.5 billion reads)

- **CPU / Threads**: ≥32 CPU
- **RAM:** ≥50 Gb RAM
- **Storage**: ≥1.5 Tb

Note: For large datasets, storage can be reduced by removing intermediate BAMs after counting.

Required Software – Pre-Processing

Software	Suggested Version	Description	Optional
STARSolo	2.7.11b	Aligns reads and generates gene-cell count matrices for single-cell RNA-seq datasets.	No
samtools	1.20	Enables viewing, sorting, indexing, and manipulation of SAM/BAM alignment files.	No
fqtk	0.3.1	Supports FASTQ file processing tasks such as filtering and barcode handling.	Yes
bash	4.0	Command-line interpreter and scripting language used to interact with an operating system	No
fastQC	0.12.1	Performs quality assessment of raw sequencing reads using summary statistics and visual reports.	Yes
RSeQC	≥5.0.4	Evaluates RNA-seq experiment quality and generates transcriptomic QC metrics.	Yes
Python	≥3.10	Provides the runtime environment for executing analysis scripts and pipeline utilities.	Yes
pandas	≥2.3.3	Provides utilities for data manipulation, such as wrapping arrays into DataFrames for CSV output.	Yes
scipy	≥1.15.3	Provides utilities for handling the sparse Matrix Market (.mtx) files produced by STARsolo.	Yes

Note: Please refer to the official webpages of each software tool mentioned to review system requirements before installation.

Pipeline Workflow

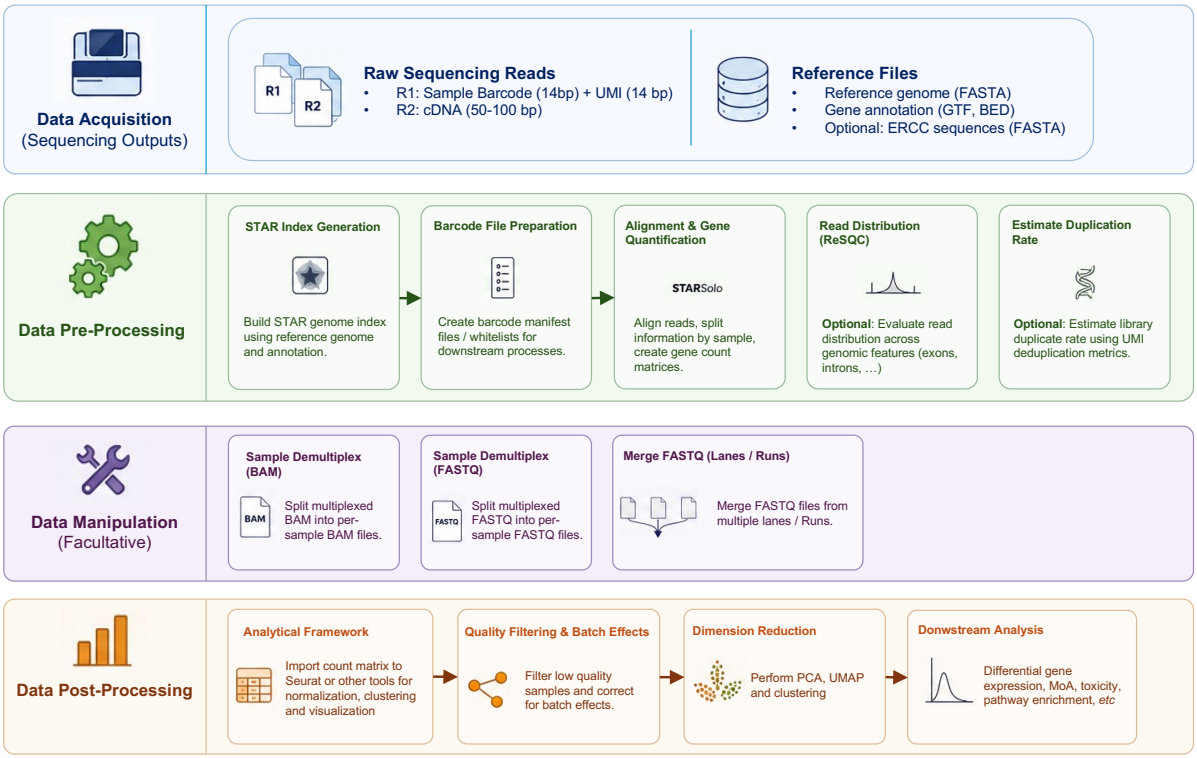

Figure 2 Schematic illustration of the pipeline workflow

Command line 1-0-1

The following figure depicts the anatomy of the command lines that will be used in this user guide.

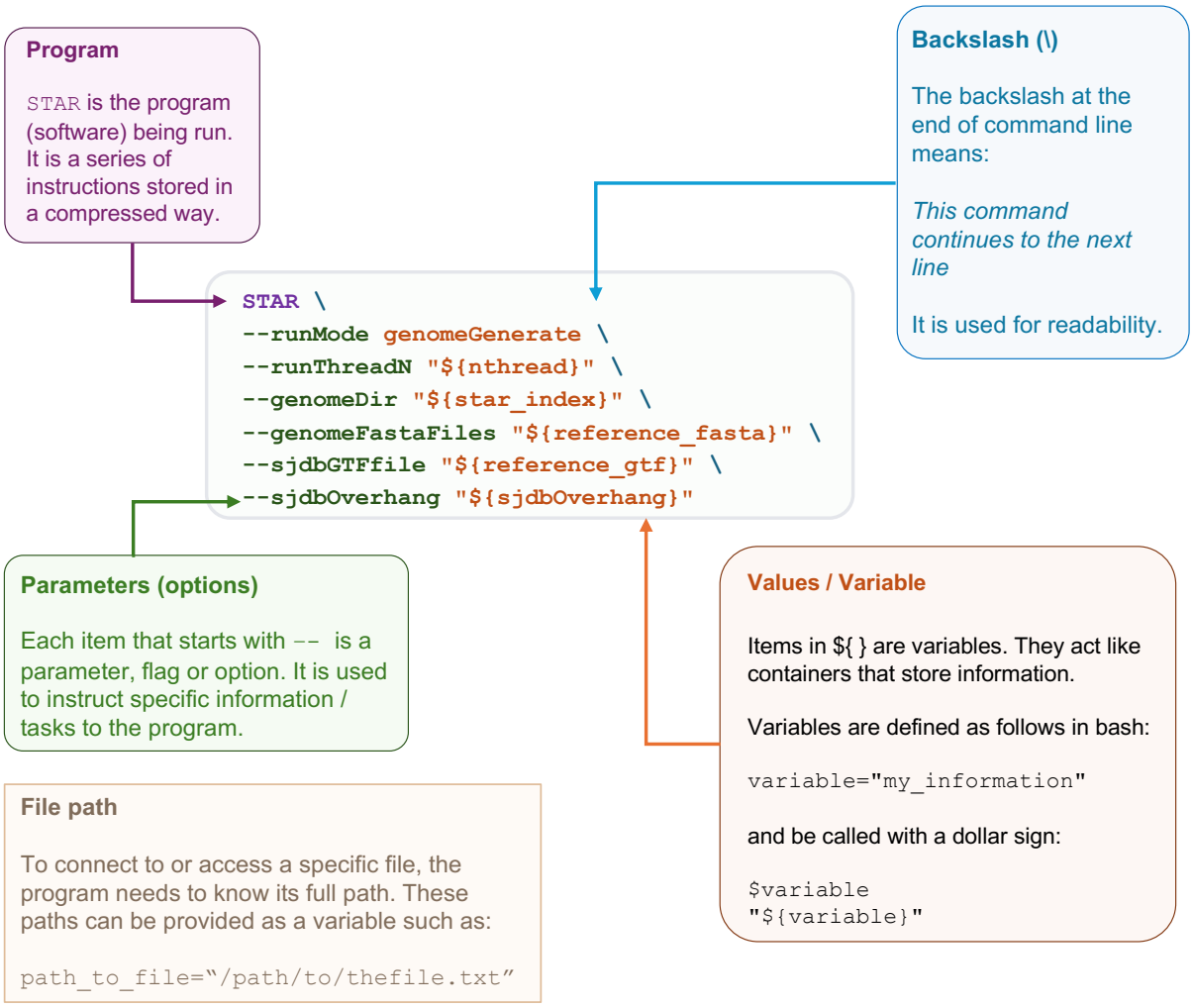

Figure 3 Schematic illustration of a bash command line.

Data Preparation

The data preparation phase establishes all reference files and metadata required for accurate MERCURIUS™ DRUG-seq processing. This includes generation of the STAR genome index, preparation of barcode whitelist and optional assessment of raw FASTQ sequencing quality. These steps ensure compatibility between sequencing reads, genome annotations, and sample barcodes prior to alignment and quantification, providing the foundation for reliable downstream transcriptomic analysis.

1.1. Building a genome index (one-time)

Overview

Alignment is the process of matching sequencing reads to their corresponding locations in a reference genome to determine their genomic origin and quantify gene expression.

This action typically requires a precomputed genome index, which is a processed and searchable version of the reference genome generated in advance to enable fast and efficient read alignment.

Because building this index requires scanning the entire genome and constructing large lookup tables, it is computationally intensive and typically performed once per genome assembly. Once generated, the indexed genome is stored and reused across multiple analyses, ensuring that subsequent alignments are much faster and more efficient.

Importantly, if ERCC RNA Spike-Ins are used, their sequences should be included in the reference genome.

Note: STAR can use up to 32-40Gb of RAM depending on the genome assembly.

Inputs

Creating the STAR index files requires two reference files:

- **GTF (Gene Transfer Format)**: provides gene and transcript annotations (exon/intron coordinates, gene models).
- **FASTA**: contains the reference genome nucleotide sequence.

These files can be downloaded from reputable repositories such as:

- **Ensembl** (https://www.ensembl.org/info/data/ftp/index.html)
- **UCSC Genome Browser** (https://hgdownload.gi.ucsc.edu/downloads.html)

The download can also be automated as exemplified here:

```
wget https://ftp.ensembl.org/pub/release-
108/fasta/homo_sapiens/dna/Homo_sapiens.GRCh38.dna.primary_assembly.fa.gz
gzip -d Homo_sapiens.GRCh38.dna.primary_assembly.fa.gz # unzip
wget htpps://ftp.ensembl.org/pub/release-
108/gtf/homo_sapiens/Homo_sapiens.GRCh38.108.gtf.gz
gzip -d Homo_sapiens.GRCh38.108.gtf.gz # unzip
```

Note: Both the FASTA and GTF are typically downloaded in compressed format (i.e. gzipped) and must be decompressed prior to next step.

FASTA and GTF selection

It is recommended to download the **primary assembly FASTA** file whenever available (i.e., the standard assembly without "_sm" or "_rm" masking tags). If the primary assembly is not provided, the **top-level assembly** should be used as an alternative.

For the **GTF annotation**, prefer the standard gene annotation file that excludes the "chr"- or "ab initio" suffixes, as these may introduce inconsistencies with downstream alignment and quantification workflows.

When using the **UCSC Genome Browser** UCSC Genome Browser, ensure consistency in chromosome naming conventions (e.g., "chr1" vs "1") between the FASTA and GTF files, as mismatches can lead to alignment or annotation errors.

Optional: ERCC FASTA sequences can be found at https://assets.thermofisher.com/TFS-Assets/LSG/manuals/ERCC92.zip.

CRITICAL: Genomes must be properly annotated and well assembled. Lower quality genomes may impact the alignment and gene detection rates or even result in a pipeline crash. It is critical to ensure that:

- The same chromosome names are present in both GTF and FASTA files
- The GTF column 3 includes the `exon` and `gene` annotation labels
- The 3' UTR of the genes are properly annotated
- The GTF contains a "`gene_id"` and `"gene_name"` information, ideally in the Ensembl/GENCODE' style

Scripts

Where to run? Bash environment

Optional: Combine the FASTA ERCC and genome sequences.

```
reference_fasta="/path/to/reference.fasta"
ercc_fasta="/path/to/ercc.fasta"
reference_ercc_fasta="/path/to/reference.gtf"
cat "${reference_fasta}" "${ ercc_fasta}" > "${reference_ercc_fasta}"
```

To generate the genome index, first define the variables:

```
star_index="/path/to/genomeDir"
reference_fasta="/path/to/reference.fasta" OR "${reference_ercc_fasta}"
reference_gtf="/path/to/reference.gtf"
nthread=8
```

The following variable may change depending on the sequencing length of the Read 2. It should be set to the read 2 length minus 1

```
sjdbOverhang=99
```

Using the following command line, run STAR to create the reference index:

```
STAR \
--runMode genomeGenerate \
--runThreadN "${nthread}" \
--genomeDir "${star_index}" \
--genomeFastaFiles "${reference_fasta}" \
--sjdbGTFfile "${reference_gtf}" \
--sjdbOverhang "${sjdbOverhang}"
```

- `--runMode:` specifies the operation mode of STAR (e.g., genome generation or read alignment).
- `--runThreadN:` number of CPU threads used to parallelize the computation.
- `--genomeDir:` path to the directory containing the STAR genome index files.
- `--genomeFastaFiles:` path to the reference genome FASTA file used to build the index.

- `--sjdbGTFfile`: path to the GTF annotation file used to supply splice junction information during indexing.
- `--sjdbOverhang`: read length minus 1, used to optimize splice junction detection during alignment.

Note: Large genomes (e.g. some plant genomes) may require some adaptations to this command line:

```
--limitSjdbInsertNsj 2000000 --limitGenomeGenerateRAM=50000000000
```

Outputs

This command line creates a new directory as defined in *`--genomeDir`*. This directory contains the index files that are used as reference for the read alignment (see 1.4).

```
star_index/
|
├──Genome
├──SA
├──SAindex
|
├──chrLength.txt
├──chrName.txt
├──chrStart.txt
|
├──GenomeInfo.txt
├──genomeParameters.txt
|
├──sjdbList.fromGTF.out.tab
|
├──Log.out
├──Log.progress.out
└── Log.final.out
```

- `Genome`: compressed binary representation of the genome.
- `SA / SAindex`: suffix array index used for fast search.
- `GenomeInfo.txt`: metadata about genome structure (chromosomes, lengths).
- `chrName.txt / chrLength.txt / chrStart.txt`: chromosome naming / coordinate
- `genomeParameters.txt`: records the parameters used to build the index.
- `sjdbList.fromGTF.out.tab`: splice junctions extracted from annotation.
- `Log.out / Log.progress.out`: main processing log.
- `Log.final.out`: summary of indexing run.

Typically, no additional quality control or validation of these files is required if the command completes successfully without errors. However, the `genomeParameters.txt` file can be inspected to retrieve a record of the exact command-line settings used to generate the index, providing useful provenance and reproducibility information.

1.2. Creating barcode files (Mandatory)

Overview

MERCURIUS™ DRUG-seq is a high-throughput RNA-seq protocol designed for large-scale gene expression profiling. It incorporates molecular barcoding at an early stage of library preparation and enables early sample pooling, thereby reducing experimental cost and increasing throughput while maintaining transcriptome-wide quantification accuracy.

To process raw sequencing data at the sample level, barcode information must be provided to define how reads are assigned to individual samples. The following script takes a text file mapping barcodes to user-defined sample names and reformats it into structured output files required for downstream analysis.

Inputs

First, create a tab-delimited **manifest** file containing two columns named `Sample_ID` and `Barcode`. Associate each sample name with its corresponding barcode from the MERCURIUS™ DRUG-seq kit. The full barcode list for your kit can be obtained from your local vendor.

```
Sample_ID     Barcode
Sample_001    ACGTGCTAGTACGA
Sample_002    TGCAACGTTGGTCA
Sample_003    GGTTACCAGTCTAA
Sample_004    CTAGGTTACGATCG
```

Notes:

- Barcode sequences must be normalized to uppercase (ACGT), in 5'->3' orientation.

- No duplicate sample IDs, no slash characters in sample names, no extra columns, and no empty rows.

Scripts

Where to run? Bash environment

Run the following command to create the barcode files necessary to process the libraries:

```
./prepare_barcodes.sh \
-i /path/to/barcodes.txt \
-w /path/to/whitelist.txt \
-f /path/to/fastq_sample_metadata.tsv \
-b /path/to/bam_barcodes.tsv \
-H
```

Where `prepare_barcodes.sh` is a bash script that can be found in 0.

- `-i`: manifest file created above.
- `-w`: output path for the STAR barcode whitelist.
- `-f`: output path for the FASTQ demux metadata TSV.
- `-b`: output path for the BAM demux barcode map.
- `-H`: optional parameter indicating whether the input manifest includes a header

Outputs

- `barcode_whitelist.txt`: One barcode per line, used by STARsolo as whitelist for cell barcode filtering.
- `fastq_sample_metadata.tsv`: Headered TSV (sample_id, barcode) used for FASTQ-level demultiplexing.
- `bam_barcodes.tsv`: Two-column sample-to-barcode mapping file used for BAM-level demultiplexing.

1.3. FASTQ Quality Assessment (Optional)

Overview

FastQC is a widely used quality control tool for high-throughput sequencing data. It analyzes raw FASTQ files and generates summary reports describing key sequencing quality metrics at the library-level, including per-base sequence quality, GC content, sequence duplication levels, adapter contamination, overrepresented sequences, and read length distribution. FastQC is typically used as an initial QC step in RNA-seq and other NGS workflows to identify potential technical issues before downstream alignment and quantification analyses.

Alternatively, MultiQC (https://github.com/multiqc/multiqc) can be used instead of FASTQC when handling multiple libraries / FASTQ.

Inputs

FASTQC takes as input the FASTQ files (read 1 and/or read 2), compressed (i.e. *.gz*) or not.

Scripts

Where to run? Bash environment

```
fastqc \
--threads 2 \
--outdir /path/to/fastqc_results/ \
/path/to/library_R1.fastq.gz /path/to/library_R2.fastq.gz
```

- `--threads`: Number of CPU threads for execution.
- `--outdir`: Writes output reports to the provided directory.
- `library_R1.fastq.gz`: Input FASTQ file containing Read 1 sequencing data.
- `library_R2.fastq.gz`: Input FASTQ file containing Read 2 sequencing data.

Outputs

FASTQC outputs the following files:

- `library_fastqc.html`: Interactive HTML report summarizing sequencing quality metrics with plots and pass/warn/fail flags.
- `library_fastqc.zip`: Compressed archive containing all raw FastQC results and data files. The ZIP archive typically includes:
  - `fastqc_data.txt`: Detailed numerical results for all QC modules.
  - `summary.txt`: PASS/WARN/FAIL status for each QC metric.
  - `fastqc_report.html`: Copy of the HTML report.
  - `Images/`: PNG images of all QC plots generated by FastQC.
  - `Icons/`: Icons used in the HTML report.

See FASTQC documentation for a more detailed overview of the QC metrics (fastqc babraham documentation).

Interpretation

Although none of these metrics should be interpreted as strict thresholds, they can provide useful indicators of reduced library or sample quality and may highlight potential technical issues requiring further inspection or preprocessing. Importantly, when processing large batches, outlier values in one or more libraries can indicate the presence of batch effects rather than true biological variation. Such deviations should therefore be interpreted in the context of the full dataset and investigated carefully to distinguish technical artifacts from genuine library-specific differences.

We recommend focusing on the following FastQC modules.

- **Per base sequence quality (Fig. 4/5a):** This plot summarizes the distribution of Phred quality scores across all sequencing cycles. Phred scores reflect the probability of incorrect base calling during sequencing. For high-quality libraries, most bases are expected to remain within the green zone throughout the read length, indicating reliable sequencing performance. A moderate decline in quality toward the end of reads is commonly observed in Illumina sequencing and is generally acceptable if overall scores remain high.
- **Per base sequence content (Fig. 4/5b):** This metric represents the proportion of each nucleotide at every sequencing cycle. In MERCURIUS™ DRUG-seq libraries, deviations from a balanced nucleotide

distribution at the beginning of reads are expected and usually do not indicate poor-quality data. Afterwards, the ratio of each nucleotide is expected to remain stable.

- **Read 1**: the first bases often correspond to cell/sample barcodes and UMIs, which are expected to be skewed (i.e. sample barcode).
- **Read 2**: early-cycle sequence bias can arise from assay-specific library construction or sequencing steps. These patterns are therefore considered normal provided they are restricted to the first cycles and remain consistent across libraries.

- **Adapter content (Fig. 4/5c):** Adapter-derived sequences should be minimal or absent in properly sized libraries. Elevated adapter content may indicate short insert sizes, over-sequencing of small fragments, or suboptimal library preparation quality. Excessive adapter contamination can interfere with alignment and downstream quantification and may require additional trimming prior to analysis. Such trimming is already included in the recommended read alignment step.
- **Overrepresented sequences (Fig. 4/5d):** Overrepresented sequences correspond to reads occurring at unexpectedly high frequencies. In DRUG-seq datasets, low levels of recurrent sequences are commonly observed and may originate from highly abundant transcripts (e.g. TTN gene in muscle cells), polyA-derived artifacts, primer-related sequences, or residual ribosomal RNA (rRNA). However, strongly enriched sequences (>3%) may indicate reduced library complexity, RNA degradation, amplification bias, or external contamination (e.g. bacterial contamination). Investigation of these sequences using alignment or BLAST-based approaches is recommended when enrichment is substantial or inconsistent across libraries.

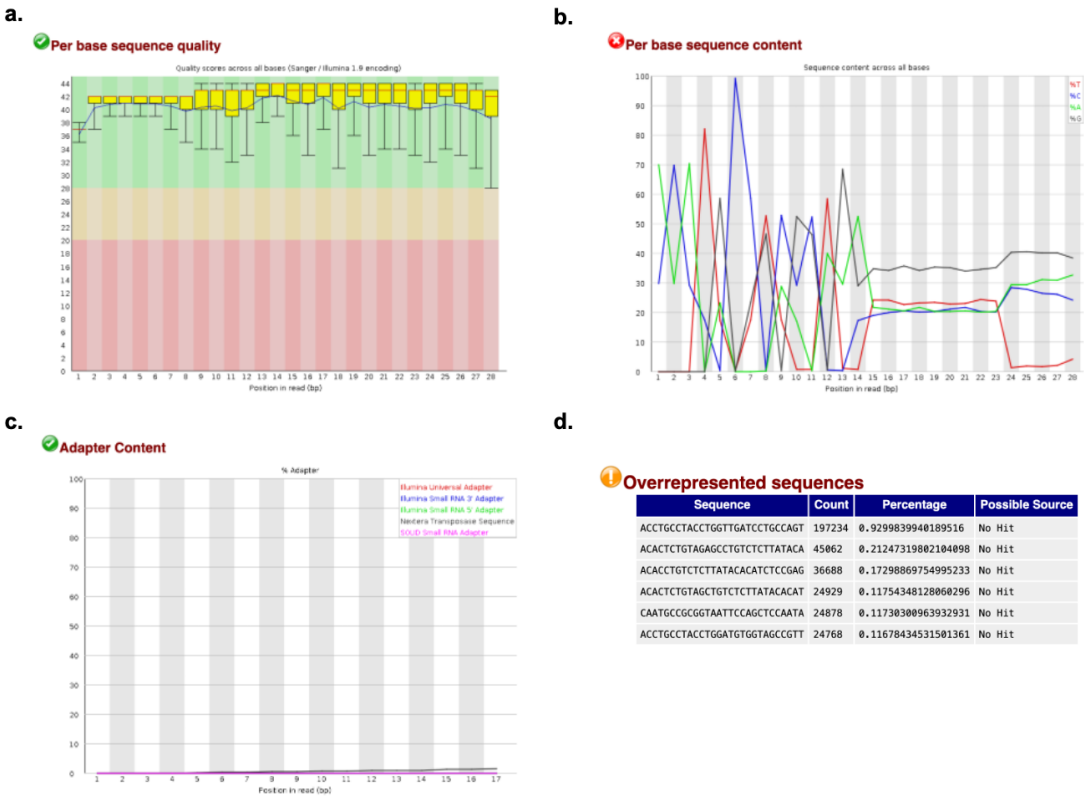

Figure 4: FASTQC results of MERCURIUS™ DRUG-seq read 1 passing QC.

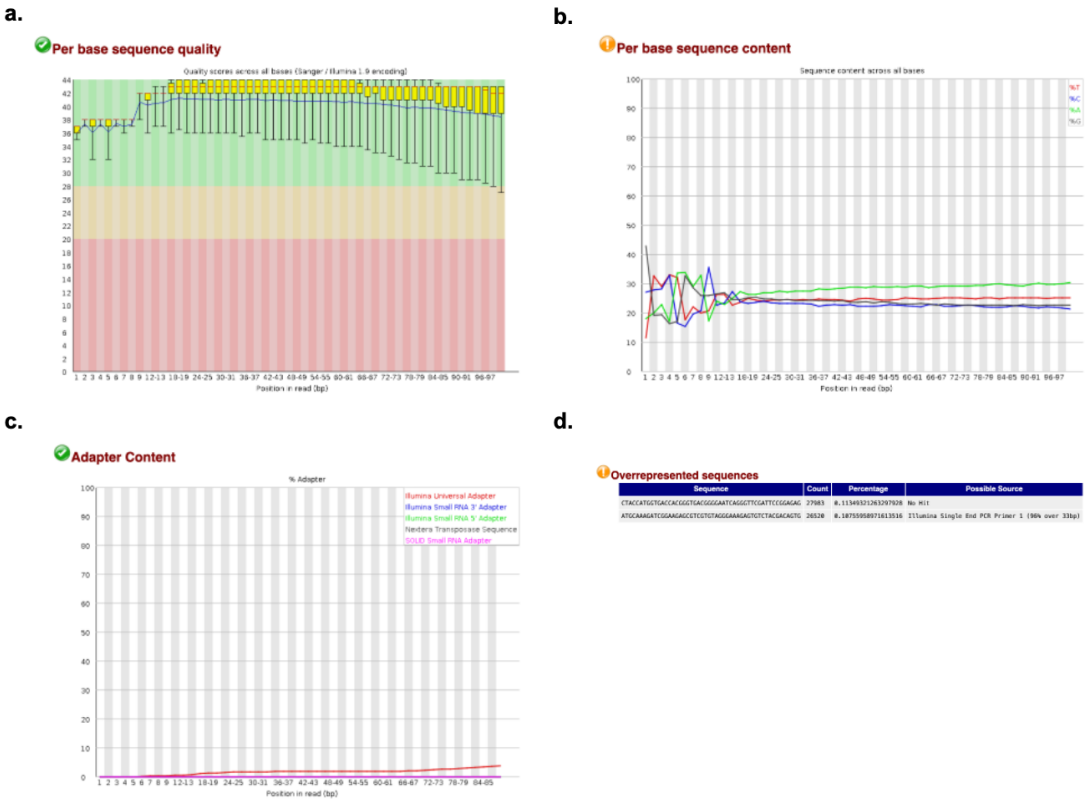

Figure 5: FASTQC results of MERCURIUS™ DRUG-seq read 2 passing QC.

Data Pre-Processing (Mandatory)

The MERCURIUS™ DRUG-seq data processing workflow converts raw sequencing reads into sample-resolved gene expression matrices suitable for downstream transcriptomic analysis. The pipeline includes reference genome preparation, barcode processing, read quality assessment, alignment, and UMI-based quantification using STARsolo, followed by optional quality control and demultiplexing procedures. Together, these steps ensure accurate assignment of sequencing reads to individual samples, robust transcript quantification, and generation of standardized outputs for downstream analyses such as normalization, clustering, dimensionality reduction, and differential expression analysis.

1.4. Alignment to the Reference Genome and Generation of UMI Count Matrix

Overview

STARsolo is a module of STAR designed for the analysis of barcoded RNA-seq data, including high-throughput protocols such as MERCURIUS™ DRUG-seq. In this workflow, sequencing reads are aligned to a reference genome while barcode sequences are simultaneously used to assign reads to their corresponding samples or cells. The command-line script below automates this process by validating inputs, preparing FASTQ files, normalizing barcode whitelist files, running STARsolo alignment, generating count matrices, and producing indexed BAM alignment files for downstream analysis.

Inputs

STARsolo alignment command requires several input files and parameters that define what data should be processed and how the alignment should be performed.

The **raw read FASTQ file** contains the RNA sequencing reads (`readFilesIn`) corresponding to raw read sequences generated by sequencing. These reads are aligned to the reference genome to identify the genes from which they originate. The following command line expects zipped files (i.e. fastq.R1.fq.gz). In this setup:

- **R1 (Read 1)** contains the **cell barcode and unique molecular identifier information** used to identify individual wells/cells and unique molecules. They are both 14 bp in length for a total read length of 28 bp (`${barcode_read}`).
- **R2 (Read 2)** contains the **genomic/cDNA sequence** derived from the captured transcript and is used for alignment to the reference genome (`${genomic_read}`). The length of this read is variable and commonly ranges from 50 to 100 bp (recommended).

The **STAR genome index directory** contains the precomputed genome reference files generated during the STAR indexing step. These files allow rapid alignment of sequencing reads to the reference genome.

To separate the samples based on their barcodes (=demultiplex), a **barcode whitelist file** containing the barcode sequences associated with each sample is provided (`soloCBwhitelist`). This file is generated in 1.2.

The `outFileNamePrefix` should contain the full output path and a prefix, that will be appended to the output files.

Importantly, this command line will only **select read mapping to a unique position** in the genome (`outFilterMultimapNmax`) for the downstream analysis. This decreases the background noise brought by multi-mapped reads such as rRNA which are then classified as “unmapped”.

Finally, STARSolo also requires parameters specifying the **positions of the sample barcode and UMI** within the reads. These parameters are specific to the MERCURIUS™ DRUG-seq protocol and should therefore not be modified (i.e. `soloCBlen`, `soloUMIstart`, and `soloUMIlen`).

As MERCURIUS™ DRUG-seq is a **stranded protocol**, meaning that the read orientations match the orientation of the corresponding genes, the `soloStrand` parameter can also be set to `"Forward"`.

Scripts

Where to run? Bash environment

Define the following variables:

```
threads=20
star_index="/path/to/star_index/genome_prefix"
barcode_read="/path/to/read1.fastq.gz"
genomic_read="/path/to/read2.fastq.gz"
barcode_whitelist="/path/to/barcode_whitelist.tsv"
output_path_prefix="/path/to/output/library_prefix_name"
```

Align the read to the reference genome and create the count matrix using this command:

```
STAR \
--runMode alignReads \
--runThreadN "${threads}" \
--genomeDir "${star_index}" \
--readFilesIn "${genomic_read}" "${barcode_read}" \
--soloCBwhitelist "${barcode_whitelist}" \
--outFileNamePrefix "${output_path_prefix}" \
--limitBAMsortRAM  1000000000  \
--outFilterMultimapNmax  1  \
--clipAdapterType  CellRanger4  \
--quantMode  GeneCounts  \
--soloCBlen  14  \
--soloUMIstart  15  \
--soloUMIlen  14  \
--soloUMIdedup  1MM_Directional  \
--soloCBmatchWLtype  1MM  \
--soloStrand  Forward  \
--soloType  CB_UMI_Simple  \
--soloCBstart  1  \
--soloBarcodeReadLength  0  \
--soloCellFilter  None  \
--outSAMmapqUnique  60  \
--outSAMunmapped  Within  \
--outBAMsortingThreadN  1  \
--outSAMattributes  NH  HI  nM  AS  CR  UR  CB  UB  GX  GN  sS  sQ  sM  \
--outSAMtype  BAM  SortedByCoordinate  \
--outSAMmultNmax  1  \
--outFilterIntronMotifs  None  \
--outBAMsortingBinsN  4  \
--readFilesCommand  zcat
```

- `--runMode alignReads`: Sets STAR to alignment mode for read mapping.
- `--runThreadN`: Number of threads used by STAR for alignment. Can be set to the number of available cores on the server node.
- `--genomeDir`: Path to the STAR genome index directory.
- `--readFilesIn "${genomic_read}" "${barcode_read}"`: Input FASTQ files (R2 genomic + R1 barcode reads).
- `--soloCBwhitelist`: Whitelist file for valid cell barcodes prepared in 1.2.
- `--outFileNamePrefix`: Output directory and prefix for all STAR results.
- `--limitBAMsortRAM`: Maximum RAM (bytes) allocated for BAM sorting.
- `--outFilterMultimapNmax`: Maximum number of allowed multiple alignments per read in count matrices.
- `--clipAdapterType`: Specifies the adapter-clipping method used to remove sequencing adapter contamination from reads before alignment.

- `--soloCBmatchWLtype`: Defines how cell barcodes are matched and corrected against the supplied whitelist (e.g., allowing sequencing-error correction).
- `--soloType`: Specifies the STARsolo processing mode and barcode/UMI layout (e.g., `CB_UMI_Simple`, `CB_UMI_Complex`, `Droplet`).
- `--soloCBstart`: Specifies the starting base position of the cell barcode sequence within the read.
- `--soloBarcodeReadLength`: Controls whether the full barcode read length is enforced or variable-length barcode reads are allowed.
- `--soloCellFilter`: Defines the algorithm and thresholds used to distinguish true cells from background/empty droplets.
- `--outSAMmapqUnique`: Sets the MAPQ (mapping quality) score assigned to uniquely mapped reads.
- `--outSAMunmapped`: Controls whether and how unmapped reads are included in the output SAM/BAM file.
- `--outBAMsortingThreadN`: Specifies the number of threads used for coordinate sorting of BAM output files.
- `--outSAMattributes`: Defines which alignment attributes/tags (e.g., NH, HI, AS, CB, UB) are included in the SAM/BAM output.
- `--outFilterIntronMotifs`: Filters alignments based on splice junction motifs to remove non-canonical or problematic introns.
- `--outBAMsortingBinsN`: Sets the number of bins used during BAM sorting, affecting memory usage and sorting performance.
- `--quantMode`: Generate a gene-by-cell count matrix by assigning reads to genes and counting UMIs per gene.
- `--soloCBlen`: Specifies the cell barcode sequence length.
- `--soloUMIstart`: Specifies the UMI start position in the barcode read.
- `--soloUMIlen`: Specifies the UMI sequence length.
- `--soloUMIdedup 1MM_Directional`: Collapse UMIs differing by up to one mismatch using a directional error-correction algorithm to reduce sequencing/PCR errors.
- `--soloStrand Forward`: Count reads assuming transcripts are sequenced in the forward strand orientation relative to annotated genes.
- `--outSAMtype BAM SortedByCoordinate`: Output alignments as a coordinate-sorted BAM file.
- `--outSAMmultNmax`: Number of alignments to report for multimapping reads (instead of all possible alignments).
- `--readFilesCommand`: Method to decompress the FASTQ files.

Outputs

STARSolo will generate both standard STAR alignment and quantification outputs.

- `${prefix}.bam`: Read alignments mapped to the reference genome (sorted).
- `Log.out`: STAR run log including all parameters used.
- `Log.progress.out`: Real-time alignment progress and mapping statistics.
- `Log.final.out`: Final summary statistics for mapping quality and alignment rates.
- `ReadsPerGene.out.tab`: Gene-level bulk read counts generated by `--quantMode GeneCounts`.
- `SJ.out.tab`: Detected splice junctions supported by aligned reads.
- Within the `Solo.out/` directory:
    - `Gene/`
        - `raw/`: Raw, unfiltered gene expression matrix output.
            - `barcodes.tsv`: List of detected cell/sample barcodes passing filtering.
            - `features.tsv`: List of quantified genomic features (typically genes).
            - `matrix.mtx`: Gene-by-sample count matrix.
        - `Features.stats`: Summary of genomic features-level count statistics.
        - `Summary.csv`: Summary statistics for cell detection and UMI/read assignment.

- `Barcodes.stats`: Statistics on barcode matching and whitelist correction.

The most important ones are the `${prefix}.bam`, `Log.final.out` and the files under `Gene/raw/`.

The first one is a compressed binary alignment file (BAM format) containing all sequencing reads aligned to the reference genome and sorted by genomic coordinates. Each entry in the file stores information such as read sequence, mapping position / quality, *etc*. It can be used for downstream QC analysis and visualize the read alignment using tools such as Integrated Genome Viewer (IGV, Broad Institute). The second file summarizes the mapping statistics. The folder contains the count matrix, using the deduplicated UMI information, and associated metadata.

Note: For long-term storage, BAM files can take up a lot of space, and we typically recommend deleting them, as well as compressing all count matrices.

Interpretation

Exploring alignment metrics generated by STAR provides valuable information for assessing overall sample and library quality (see Figure 3). The `Log.final.out` file summarizes key sequencing and mapping statistics, including the percentage of uniquely mapped reads, multimapped reads, unmapped reads, mismatch rates, splice junction detection, and barcode-processing metrics.

These metrics can vary substantially across tissues, cell types, and experimental conditions. Their interpretation should therefore be performed within the broader context of the experiment and evaluated comparatively across samples processed under the same conditions.

- **Uniquely mapped reads:** Main metrics, presenting the percentage of reads mapping to a unique position. A high proportion of uniquely mapped reads generally reflects good sequencing quality and a complex library. Lower mapping rates may result from poor RNA quality, contaminations, excessive adapter content, incomplete reference annotation, or sequencing artifacts. This value should typically be around 60-85% of the MERCURIUS™ DRUG-seq libraries.
- **Multimapped reads (Too many loci >1):** Reads mapping to multiple genomic locations are commonly associated with repetitive regions, homologous genes, pseudogenes, or residual ribosomal RNA. Moderate levels are expected in transcriptomic datasets, but excessive multimapping may indicate low library complexity or contamination.
- **Unmapped reads:** The proportion of unmapped reads should remain limited. These often regroup adapter sequences, reads mapping to too many loci (>10), lower quality or damaged sequences, bacteria contaminations, *etc*.

Note: An unexpectedly high proportion of unmapped reads can also indicate an error in the selected genome (i.e. mapping human data on rat genome).

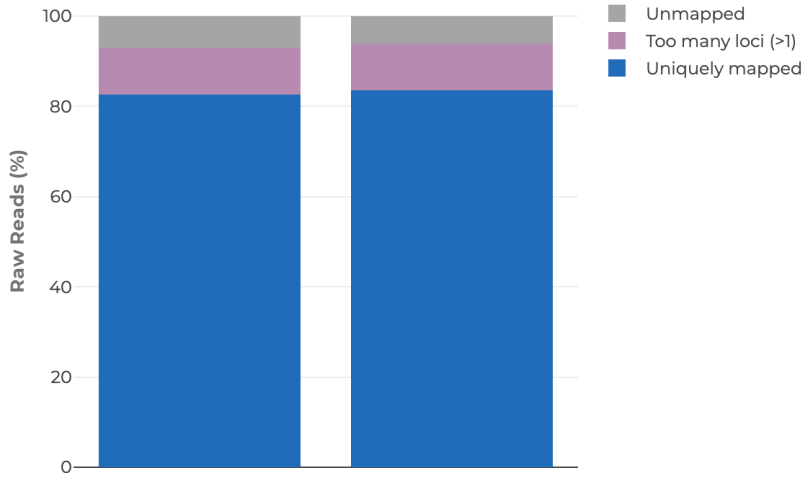

Figure 6: Read Alignment Summary for two MERCURIUS™ successful libraries.

Optional Pre-Processing

1.5. Read Distribution with RSeQC

Overview

RSeQC (Read Quality Control) is a preprocessing and assessment package used in sequencing pipelines to evaluate the sample quality before downstream analysis.

In the context of MERCURIUS™ DRUG-seq, it is primarily used to assess how mapped reads are distributed across the genome, distinguishing between exonic, intronic, and intergenic alignments by leveraging read tags and alignment annotations to evaluate library composition and transcriptional signal quality.

Inputs

RSeQC requires a **BAM** (i.e. *.bam*) file along with its corresponding index (i.e. *.bam.bai*). The index creation is described below using SAMtools, a software package for parsing and manipulating alignments in SAM/BAM formats.

In addition, it requires a BED (Browser Extensible Data) file, a plain-text format used to represent genomic intervals. Each line typically defines a genomic region with at least chromosome, start, and end coordinates. Importantly, for RSeQC, the BED file must be in **BED12** format, containing 12 columns that encode exon-level gene structure and enable accurate read distribution and gene body coverage analyses. Such file can be obtained from different sources:

- **RSeQC website (Recommeded):** Pre-computed BED12 gene models (https://rseqc.sourceforge.net/).
- **Converting a GTF file:** into BED12 using UCSC utilities

```
gtfToGenePred annotations.gtf annotations.genePred
genePredToBed annotations.genePred annotations.bed
```

Scripts

Where to run? Bash environment

To generate the genome index, first define the variables:

```
bam_file=/path/to/Aligned.sortedByCoord.out.bam
bed_file=/path/to/reference.bed
output_file=/path/to/library_name.distribution.txt
samtools_nthreads=8
```

Then run the following to index the BAM file. This step is necessary to allow for fast and efficient computing by RSeQC. It creates a `"${bam_file}".bai` (=index) file in the same folder as the BAM file.

```
samtools index "${bam_file}" -@ "${samtools_nthreads}"
```

Finally, use this command line to estimate the read distribution across the genomic features:

```
read_distribution.py -i "${bam_file}" -r "${bed_file}" > "${output_file}"
```

- `-i`: Input BAM file containing aligned sequencing reads.
- `-r`: Reference BED12 file defining genomic gene structures used for read distribution analysis.
- `> "${output_file}"`: Redirects the output report (read distribution statistics) to a specified file.

Outputs

SAMtools will create a `Aligned.sortedByCoord.out.bam.bai` file.

RSeQC will create a `library_name.distribution.txt` file.

Interpretation

The `library_name.distribution.txt` summarizes how the reads are distributed across the following genomic features:

- **CDS exons:** Protein-coding regions of genes that are translated into amino acids.
- **5' UTR / 3' UTR exons:** Untranslated regions of exons involved in transcript regulation rather than protein coding.
- **Introns:** Non-coding sequences within genes that are removed during RNA splicing.
- **Intergenic regions:** Genomic regions located between annotated genes.
- **TSS / TES (1, 5 or 10 kb):** Regions upstream of transcription start sites (TSS) or downstream of transcription end sites (TES) within defined distance windows (1, 5, or 10 kb).

The number of Total Assigned Tags serves as the denominator for calculating the proportion of reads assigned to each genomic feature category. This information can be used as quality control.

- Most reads should map to **CDS exons** and **3′ UTR exons** (typically ~40–60% combined), reflecting expected transcript-derived signal in a 3′-biased RNA-seq protocol.
- An elevated fraction of **intergenic reads** (>12%) may indicate genomic DNA contamination, nonspecific capture, or low library complexity (i.e. few cells).
- **Intronic reads** are more variable in interpretation: they largely originate from pre-mRNA and reflect transcriptional activity and RNA processing states, but they are also frequently enriched in lower-quality samples (e.g., partially degraded RNA, fixed material, permeabilized or dying cells), where nuclear retention or incomplete RNA maturation increases intronic signal.
- As a 3′ RNA-seq protocol, **5′ UTR, TSS, and TES regions** are not expected to be strongly enriched in MERCURIUS™ DRUG-seq data, due to capture bias toward transcript ends and polyadenylated RNA fragments.

Importantly, these distributions are highly tissue- and cell-type dependent. For example, PBMC samples often show higher intronic content compared to cell lines such as HEK293, reflecting biological differences in transcriptional dynamics and nuclear RNA abundance. While read distribution profiles are valuable indicators of library quality and potential technical artifacts, they must always be interpreted in a biological context, with careful monitoring for systematic batch effects or deviations across similar samples.

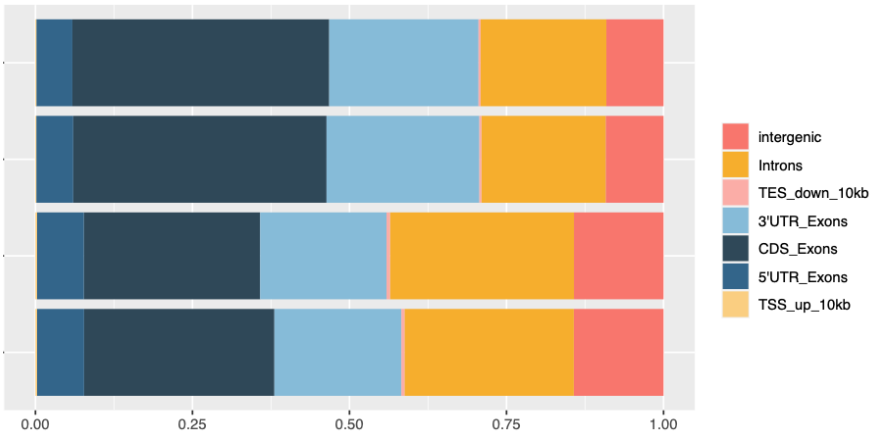

Figure 7: RSeQC Read Distribution. Ratio of the different features among mapped reads in four different libraires. Bottom two libraries were generated from a sub-optimal number of cells which led to an increase in intergenic reads.

1.6. Estimate Duplication Rate

Overview

In 3′ RNA-seq workflows, duplicate reads can arise either from true biological abundance or from PCR amplification during library preparation. Unique Molecular Identifiers (UMIs) enable these sources to be distinguished by tagging individual RNA molecules prior to amplification, allowing PCR duplicates to be identified and collapsed during data processing. Evaluating duplication rates is therefore an important quality-control step, providing insight into library complexity, sequencing depth, and the extent of amplification bias. The following section describes how duplication metrics are calculated and interpreted in UMI-based 3′ RNA-seq datasets.

Inputs

The script is based on STARsolo's raw outputs generated in section 1.4. More specifically, duplication rate is derived by comparing count matrices computed with and without UMI deduplication ("`1MM_Directional`" and "`NoDedup`" options, respectively.

In STARsolo, the deduplication method is governed by the `--star-solo-umi-dedup` parameter, which accepts a list of options so that both total reads and UMI count matrices can be obtained in a single run using for example:

```
--star-solo-umi-dedup "1MM_Directional NoDedup"
```

This creates new count matrices in the `/Gene/raw/` folder:

```
umiDedup-1MM_Directional.mtx
umiDedup-NoDedup.mtx
```

These matrices can be used for evaluating the duplication rate.

Scripts

Where to run? Bash environment

To obtain UMI duplication statistics, first define the variables:

```
star_align_dir="/path/to/star_align_dir/"
dedup_method="1MM_Directional"
output_csv_path="/path/to/duplication_statistics.csv"
```

Then run the following command:

```
python get_duplication_statistics.py \
        --raw_dir "${star_align_dir}" \
        --dedup_method "${dedup_method}" \
        --output_csv "${output_csv_path}"
```

Where `generate_umi_statistics.py` is a python script that can be found in Appendix 2.

- `--raw_dir`: input folder containing STARsolo count matrices (both UMI and non-deduplicated reads).
- `--dedup_method`: method used as `STARsolo --soloUMIdedup` argument to obtain UMI counts (e.g. '1MM_Directional').
- `--output_csv_path`: output path for generated deduplication statistics. A summary file containing total counts across all samples will also be generated and named using the "`.summary`" prefix.

Interpretation

Duplication rates in UMI-based 3′ RNA-seq reflect the proportion of sequencing reads originating from molecules that have already been observed. High duplication rates generally indicate reduced library complexity or excessive library amplification.

Several factors can contribute to increased duplication rates:

- **Low cell numbers or low RNA input**, resulting in fewer unique transcript molecules available for sequencing.
- **High sequencing depth**, where the same molecules are sampled repeatedly once library complexity is saturated.
- **Excessive library amplification**, often required when starting material is limited, which can lead to over-representation of a subset of molecules.

Of note, high duplication rates can be a useful indicator of sample quality issues. Low-quality samples or treatments causing substantial cytotoxicity may reduce the number of viable cells and the amount of recoverable RNA. This decreases library complexity and often necessitates additional PCR cycles during library preparation, both of which can contribute to elevated duplication rates.

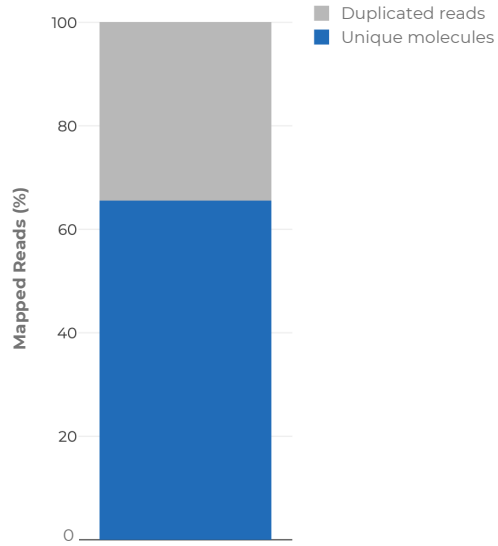

Figure 7: Duplication rate estimation in MERCURIUS™ DRUG-seq, ~65% of unique molecules sequenced.

Data Manipulation

1.7. Sample Demultiplexing (BAM) (Optional)

Overview

In MERCURIUS™ DRUG-seq workflow, BAM demultiplexing is used to split a pooled alignment file into individual sample-level BAM files based on barcode annotations embedded in the reads (e.g., well barcodes). By extracting reads according to their barcode assignments, each sample can be analyzed independently. In some specific cases, such as submitting data to online repositories (e.g. GEO or ArrayExpress) or running standard bulk RNA-seq pipelines, users may also perform BAM demultiplexing to facilitate sample-specific quality control, troubleshoot barcode assignment issues, or enable downstream analyses that require strictly separated per-sample alignment files.

Inputs

The following scrips starts from a **BAM** file generated by STARSolo (1.4). This BAM must contain a tag_name (default: `CR`) which is associated to the sample barcode.

In addition, a **barcode manifest file**, associating the barcode sequence to a specific sample is provided. This barcode file is a two-column sample/barcode map that can be obtained from 1.2 (`bam_barcodes.tsv`).

Scripts

Where to run? Bash environment

To demultiplex the BAM, first define the variables:

```
output_dir="/path/to/output_dir/"
bam_file="/path/to/input.bam"
barcode_file="/path/to/bam_barcodes.tsv"
tag_name="CR"
```

Then run the following command:

```
./demultiplex_bam.sh \
"${output_dir}" \
"${bam_file}" \
"${barcode_file}" \
"${tag_name}"
```

Where `demultiplex_bam.sh` is a bash script that can be found in Appendix 3.

- `output_dir`: directory where per-sample BAM files produced by demultiplexing will be written.
- `bam_file`: input multiplexed BAM file containing all samples combined.
- `barcode_file`: tab-delimited file mapping sample IDs to barcode values used for splitting the BAM.
- `tag_name="CR"`: BAM tag used to identify and filter reads based on barcode assignment.

Outputs

The code will create one BAM file for each demultiplexed sample in the output directory:

```
${output_dir}/${sample_id}.bam
```

1.8. Sample Demultiplexing (FASTQ) (Optional)

Overview

In MERCURIUS™ 3′ DRUG-seq, FASTQ demultiplexing is used to separate pooled sequencing output into individual sample-level FASTQ files based on barcode or index sequences embedded in Read 1 (e.g., well or plate barcodes). This process assigns each sequencing read to its correct biological sample at the earliest stage of the pipeline.

In some specific cases, such as transcript quantification analysis or running standard RNA-seq preprocessing workflows, users may perform FASTQ demultiplexing to recover per-sample raw reads, validate barcode assignment, or enable compatibility with pipelines that require strictly separated input FASTQ files.

The following example uses fqtk tool (Fulcrum Genomics, https://github.com/fulcrumgenomics/fqtk).

Inputs

The fqtk tool takes the raw **FASTQ** files (read 1 and read 2) as inputs. It uses a **barcode manifest file**, associating the barcode sequence to a specific sample ID. This barcode file is a two-column sample/barcode map that can be obtained from 1.2 (`fastq_sample_metadata.tsv`).

Reads that do not match any barcode from the manifest are placed in a “unmatched” FASTQ file.

Scripts

To demultiplex the FASTQ files, first define the variables:

```
output_dir="/path/to/output_dir/"
read1="R1.fastq.gz"
read2="R2.fastq.gz"
barcode_ref="barcodes.tsv"
r1_cigar="14B14M"
r2_cigar="90T"
threads=5
unmatched_prefix="unmatched"
```

Then create the output directory.

```
mkdir -p "${output_dir}"
```

Demultiplex the reads.

```
fqtk demux \
-i "${read1}" "${read2}" \
-r "${r1_cigar}" "${r2_cigar}" \
-s "${barcode_ref}" \
-o "${output_dir}" \
-u "${unmatched_prefix}" \
-t "${threads}"
```

- `-i`: input FASTQ files to be demultiplexed, first read 1 then read 2
- `-r`: read structure definitions for R1 and R2 specifying barcode/sequence layout
- `-s`: sample-to-barcode metadata table used for assigning reads to samples
- `-o`: output directory where demultiplexed FASTQ files are written
- `-u`: prefix used for reads that cannot be assigned to any sample
- `-t`: number of threads used for parallel execution of fqtk demultiplexing

Outputs

The code will create one BAM file for each demultiplexed sample in the output directory:

- `sampleA_R1.fq.gz`: Demultiplexed FASTQ file containing R2 reads assigned to each individual sample based on barcode matching.
- `unmatched_R1.fq.gz`: Reads that could not be assigned to any sample barcode and are therefore kept separate for QC or troubleshooting.
- `demux-metrics.txt`: Summary report describing barcode assignment efficiency, read counts per sample, and fraction of unmatched reads.

Note: By default, fqtk name the output file R1 even though the file contains R2 reads. UMI sequences are placed in the FASTQ read names.

1.9. Merging FASTQ from Multiple Sequencing Lanes / Runs (Optional)

Overview

Depending on the sequencing instrument and flow cell configuration, a single library may generate multiple R1 and R2 FASTQ files corresponding to different lanes and, in some cases, different sequencing runs. For downstream processing, FASTQ files from the same library should be merged into unique `R1.fastq.gz` and `R2.fastq.gz` files to ensure consistent and simplified input for subsequent analysis steps. Below is an example of FASTQ files generated from a HiSeq run with four lanes.

Inputs

FASTQ files split between lanes or sequencing runs

```
mylibrary_L001_R1.fastq.gz, mylibrary_L002_R1.fastq.gz,
mylibrary_L003_R1.fastq.gz, mylibrary_L004_R1.fastq.gz
mylibrary_L001_R2.fastq.gz, mylibrary_L002_R2.fastq.gz,
mylibrary_L003_R2.fastq.gz, mylibrary_L004_R2.fastq.gz
```

CRITICAL: All files should either be compressed (e.g. *.gz*) or decompressed (e.g. *.fastq*).

Scripts

The different files are concatenated using the bash function *cat*

```
cat mylibrary_L001_R1.fastq.gz mylibrary_L002_R1.fastq.gz
mylibrary_L003_R1.fastq.gz mylibrary_L004_R1.fastq.gz > mylibrary_R1.fastq.gz
cat mylibrary_L001_R2.fastq.gz mylibrary_L002_R2.fastq.gz
mylibrary_L003_R2.fastq.gz mylibrary_L004_R2.fastq.gz > mylibrary_R2.fastq.gz
```

CRITICAL: The order of merging files should be kept the same (for e.g., R1_L001 and R2_L001 first).

Outputs

- `mylibrary_R1.fastq.gz`: Concatenated FASTQ read 1 file
- `mylibrary_R2.fastq.gz`: Concatenated FASTQ read 2 file

Data Post-Processing

Post-processing of DRUG-seq RNA-seq data is primarily aimed at translating high-dimensional transcriptional profiles into **biological and pharmacological insights**. Depending on the experimental context, this analysis can support:

- **Mechanism of Action (MoA) inference**: identifying transcriptional signatures associated with specific perturbations or drug classes.
- **Toxicity and stress response profiling**: detecting generic or compound-specific stress pathways (e.g., ER stress, DNA damage, apoptosis).
- **Treatment prioritization:** comparing compounds to reference profiles to rank candidates based on efficacy and safety signals.
- **Biomarker discovery**: identifying treatment-specific or dose-responsive genes.
- **Pathway and network activity inference**: linking expression changes to functional biological processes.

A single rigid post-processing pipeline cannot be universally described because DRUG-seq studies vary widely in:

- Experimental design (i.e. number of plates, replicates, controls)
- Perturbation types (i.e. small molecules, genetic perturbations, combinations)
- Biological systems (i.e. cell lines, primary cells, mixed populations)
- Signal strength and heterogeneity of transcriptional responses

Therefore, the workflow below represents a **recommended analytical framework rather than a fixed pipeline**, intended to be adapted to dataset scale and biological context.

Analytical Framework

We recommend using **Seurat** (R) or **Scanpy** (python) as the primary analysis environments for MERCURIUS™ DRUG-seq post-processing. Both frameworks support scalable workflows for normalization, batch correction, dimensionality reduction, and differential analysis, and can be selected based on user preference or ecosystem compatibility.

Quality Controls

To evaluate the quality of the results, we recommend performing the following quality controls:

1. Number of detected genes:

Typically, >10,000 genes per sample at ~1M reads per well, though this varies with cell type, cell input number, and transcriptional activity.

2. Exonic read fraction:

Generally expected to be ~40–60%, depending on library quality and biological system.

3. Library consistency across samples (key QC criterion):

- Uniquely mapped reads
- Number of detected genes (at fixed sequencing depth)
- Exonic read percentage

Systematic deviations may indicate:

- batch effects (technical variation between plates or processing runs)
- true biological shifts (i.e. strong treatment effects)

➔ These should be distinguished through visualization and covariate-aware analysis.

4. ERCC (if included):

- ERCC expression should remain **stable across libraries under comparable conditions**.
  - **Random or isolated deviations**: may reflect compound-specific biological effects, including **toxicity or global transcriptional suppression**
  - **Systematic patterns (e.g., plate, row, or column effects)**: suggest technical issues such as:
    - inefficient lysis
    - pipetting or liquid handling errors
    - transfer artifacts during processing

Batch Correction and Data Integration

When analyzing multiple DRUG-seq plates, account for **plate-to-plate variations** as a batch effect (Figure 8).

Integrate plates before downstream analysis using one of the following approaches:

- **Explore:** Seurat integration
- **Alternative methods**: ComBat or limma batch correction

Confirm successful integration by verifying that samples cluster primarily by **biological treatment** rather than by plate.

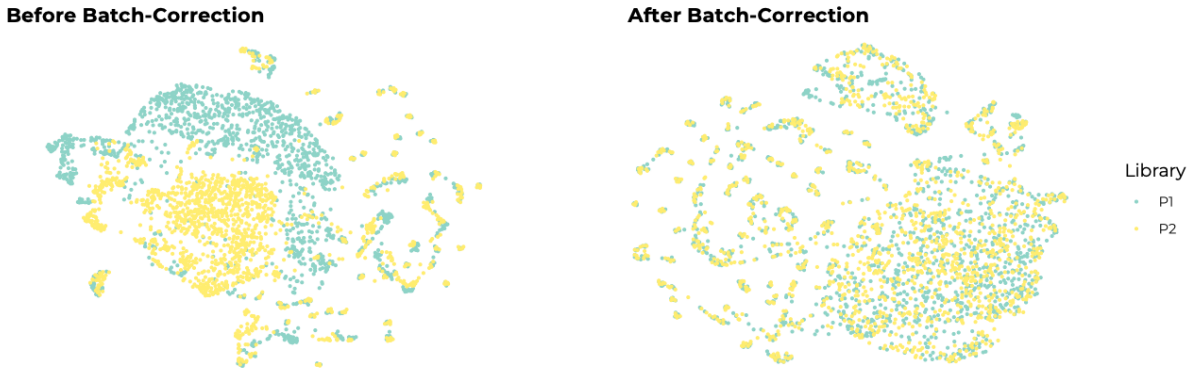

Figure 8: UMAP plot highlighting a minor batch effect between two experiments, before (left) and after (right) integration to correct for batch effect.

Differential Expression Analysis

Perform differential expression (DE) analysis using traditional DE analysis tools such as **DESeq2**.

Note: Including **plate annotation as a covariate** in the design formula can improve the results.

Use the largest feasible pool of control samples, as increased control numbers generally improve statistical power and DEG detection.

Recommended significance criteria:

- Absolute fold change ≥ 1.5
- Adjusted p-value (FDR) < 0.05

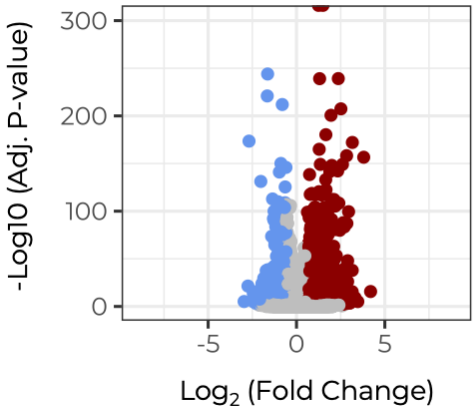

Figure 9: Volcano plot highlighting DE genes with a negative (blue) or positive (red) log-fold changes.

Dimensionality Reduction and Visualization

Generate low-dimensional representations to assess treatment effects and sample relationships.

UMAP is recommended for visualization, as it typically provides better separation of treatment groups than PCA.

Explore the first **10–30 principal components (PCs)** as input for UMAP.

Examine clustering patterns to identify treatment-specific transcriptional responses and potential outliers.

Figure 10: UMAP depicting 3072 MERCURIUS™ DRUG-seq samples treated with 33 compounds (8-doses), colored by compound.

Treatment-Specific Signature Discovery

Perform marker gene analysis for each treatment group.

Seurat marker analysis can be used to identify genes specifically enriched or depleted in individual treatments.

Use identified markers for:

- Mechanism-of-action exploration
- Treatment comparison
- Pathway and downstream biological interpretation
- Point-of-departure studies
- Effective dose

See Figure 11 and 12 below for further examples:

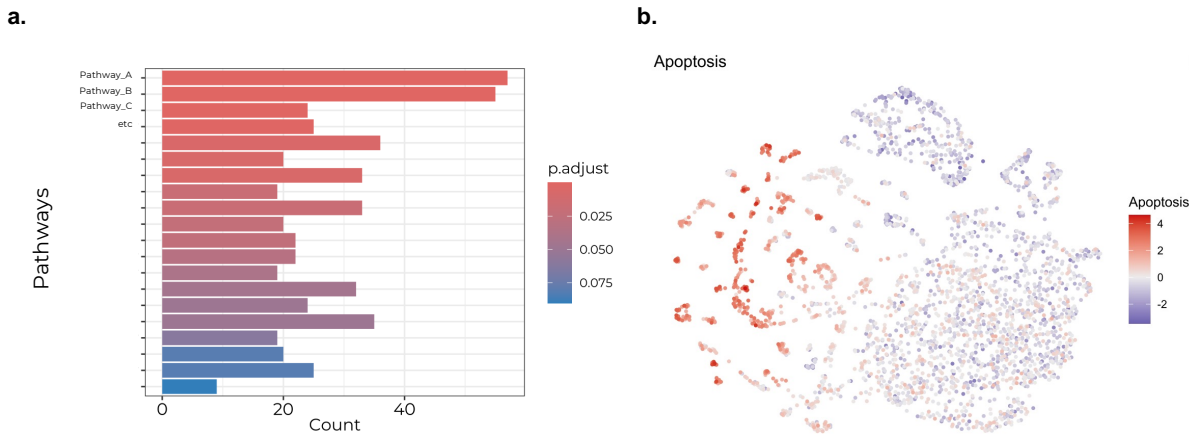

Figure 11: a. Gene set enrichment analysis results (GSEA, *clusterProfiler*), highlighting key activated pathways for a specific treatment condition. **b.** UMAP depicting the expression of genes related to apoptosis.

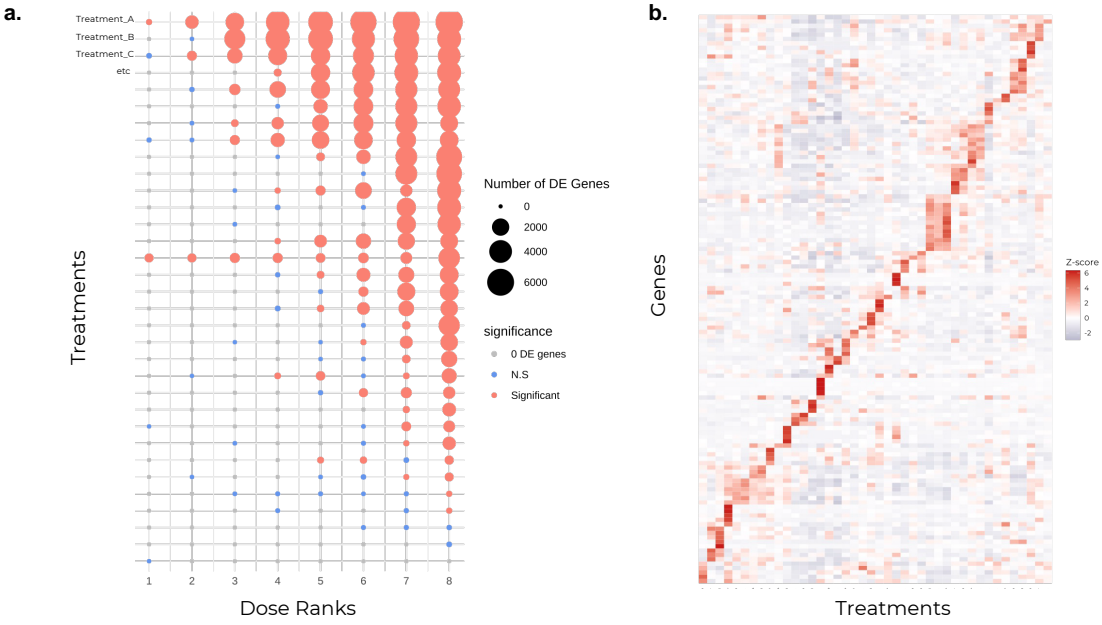

Figure 12: **a.** Graphic depicting the number of differentially expressed genes (DE) for each treatment according to the dose. Information used to define the point-of-departure (POD) **b.** Heatmap of average expressed counts. Treatments are clustered by commonly expressed genes, highlighting common pathways.

Appendix 1. prepare_barcodes.sh

Open a text editor and save the following code in a file called: `prepare_barcodes.sh`

```
#!/usr/bin/env bash
set -euo pipefail

script_name="$(basename "$0")"
input_file=""
whitelist_file=""
fastq_file=""
bam_file=""
has_header=false
declare -A seen_samples=()

usage() {
    cat <<EOF
Usage: ${script_name} -i INPUT_FILE -w WHITELIST_FILE -f FASTQ_FILE -b BAM_FILE
[-H]
Validate a strict two-column sample/barcode manifest and derive downstream
barcode files.
Required options:
  -i INPUT_FILE     Two-column manifest with sample ID in column 1 and barcode
in column 2.
  -w WHITELIST_FILE Output path for the STAR barcode whitelist.
  -f FASTQ_FILE     Output path for the FASTQ demux metadata TSV.
  -b BAM_FILE       Output path for the BAM demux barcode map.
  -H                Treat the first row as a header and skip it.
  -h                Show this help text and exit.
EOF
}

while getopts ":i:w:c:f:b:Hh" opt; do
    case "$opt" in
        i)
            input_file="$OPTARG"
            ;;
        w)
            whitelist_file="$OPTARG"
            ;;
        f)
            fastq_file="$OPTARG"
            ;;
        b)
            bam_file="$OPTARG"
            ;;
        H)
            has_header=true
            ;;
        h)
            usage
            exit 0
            ;;
        :)
            echo "Option -${OPTARG} requires a value." >&2
            usage >&2
            exit 1
            ;;
        ?)
            echo "Unknown option: -${OPTARG}" >&2
            usage >&2
            exit 1
            ;;
```

```
    esac
done
shift $((OPTIND - 1))

if [[ $# -ne 0 ]]; then
    echo "Unexpected positional arguments: $*" >&2
    usage >&2
    exit 1
fi

required_args=(
    "$input_file"
    "$whitelist_file"
    "$fastq_file"
    "$bam_file"
)
for required_arg in "${required_args[@]}"; do
    if [[ -z "$required_arg" ]]; then
        echo "All output and input paths are required." >&2
        usage >&2
        exit 1
    fi
done

declare -A seen_filenames=()
declare -A seen_filename_flags=()
path_args=(
    "-i:$input_file"
    "-w:$whitelist_file"
    "-f:$fastq_file"
    "-b:$bam_file"
)
for path_arg in "${path_args[@]}"; do
    arg_flag="${path_arg%%:*}"
    arg_path="${path_arg#*:}"
    arg_name="$(basename "$arg_path")"
    if [[ -n "${seen_filenames[$arg_name]:-}" ]]; then
        echo "Conflicting file names provided to ${script_name}: ${arg_flag} and
${seen_filename_flags[$arg_name]} both use '${arg_name}'. Input and output file
names must be unique to avoid overwriting files." >&2
        exit 1
    fi
    seen_filenames["$arg_name"]=1
    seen_filename_flags["$arg_name"]="$arg_flag"
done

if [[ ! -f "$input_file" ]]; then
    echo "Barcode manifest not found: $input_file" >&2
    exit 1
fi

: > "$whitelist_file"
printf 'sample_id\tbarcode\n' > "$fastq_file"
: > "$bam_file"

line_number=0
processed_samples=0
while IFS=$'\t' read -r sample_id barcode extra || [[ -n "$sample_id" || -n
"$barcode" || -n "$extra" ]]; do
    line_number=$((line_number + 1))
    if [[ "$has_header" == true && $line_number -eq 1 ]]; then
        continue
    fi
```

```
    sample_id="${sample_id%$'\r'}"
    barcode="${barcode%$'\r'}"
    extra="${extra%$'\r'}"
    if [[ -z "$sample_id" && -z "$barcode" && -z "$extra" ]]; then
        echo "Invalid barcode manifest at line ${line_number}: empty lines are
not allowed." >&2
        exit 1
    fi
    if [[ -n "$extra" ]]; then
        echo "Invalid barcode manifest at line ${line_number}: expected exactly
2 tab-delimited columns." >&2
        exit 1
    fi
    if [[ -z "$sample_id" || -z "$barcode" ]]; then
        echo "Invalid barcode manifest at line ${line_number}: sample ID and
barcode are required." >&2
        exit 1
    fi
    if [[ "$sample_id" == */* ]]; then
        echo "Invalid barcode manifest at line ${line_number}: sample IDs must
not contain '/'." >&2
        exit 1
    fi
    if [[ -n "${seen_samples[$sample_id]:-}" ]]; then
        echo "Invalid barcode manifest at line ${line_number}: duplicate sample
ID '${sample_id}'." >&2
        exit 1
    fi
    normalized_barcode="$(printf '%s' "$barcode" | tr '[:lower:]' '[:upper:]')"
    if [[ ! "$normalized_barcode" =~ ^[ACGTN]+$ ]]; then
        echo "Invalid barcode manifest at line ${line_number}: barcode must
contain only ACGTN characters." >&2
        exit 1
    fi
    seen_samples["$sample_id"]=1
    processed_samples=$((processed_samples + 1))
    printf '%s\n' "$normalized_barcode" >> "$whitelist_file"
    printf '%s\t%s\n' "$sample_id" "$normalized_barcode" >> "$fastq_file"
    printf '%s\t%s\n' "$sample_id" "$normalized_barcode" >> "$bam_file"
done < "$input_file"

if [[ $processed_samples -eq 0 ]]; then
    echo "Invalid barcode manifest: file is empty." >&2
    exit 1
fi
```

Ensure the script is executable by running the following command from the file's directory:

```
chmod a+x prepare_barcodes.sh
```

Appendix 2. generate_umi_statistics.py

Open a text editor and save the following code in a file called: `generate_umi_statistics.py`

```
import pandas as pd
import numpy as np
import sys
from pathlib import Path
from scipy.io import mmread
import argparse

def load_matrix(raw_dir, filename):
    return mmread(Path(raw_dir) / filename).tocsc()  # Convert to CSC format
for efficient column operations

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Calculate UMI deduplication
statistics.")
    parser.add_argument("--raw_dir", type=str, required=True, help="Directory
containing raw matrices and barcodes.")
    parser.add_argument("--dedup_method", type=str, required=True,
help="Deduplication method used (e.g., '1MM_Directional').")
    parser.add_argument("--output_csv", type=str, required=True, help="Path to
output CSV file for duplication statistics.")
    args = parser.parse_args()

    # Load barcode and feature names
    Barcodes = pd.read_csv(Path(args.raw_dir) / "barcodes.tsv",
header=None)[0].tolist()

    # Load matrices
    mat_umi   = load_matrix(args.raw_dir, f"umiDedup-{args.dedup_method}.mtx")
# deduplicated
    mat_reads = load_matrix(args.raw_dir, "umiDedup-NoDedup.mtx") # raw reads

    if mat_umi.shape != mat_reads.shape:
        sys.exit("ERROR: UMI and NoDedup matrices have different shapes.")

    # colSums — equivalent to R colSums(), summing over all genes per barcode
    umi_counts   = np.asarray(mat_umi.sum(axis=0)).flatten()    # shape:
(n_barcodes,)
    read_counts  = np.asarray(mat_reads.sum(axis=0)).flatten()

    with np.errstate(divide="ignore", invalid="ignore"):
        umi_pct = np.where(read_counts > 0, np.round(umi_counts / read_counts
* 100, 2), np.nan)

    result = pd.DataFrame({
        "sample":      barcodes,
        "umi_count":   umi_counts.astype(int),
        "read_count":  read_counts.astype(int),
        "umi_pct":     umi_pct,
    })
    summary = pd.DataFrame({
        "total_barcodes": [len(barcodes)],
        "total_umi": [umi_counts.sum()],
        "total_reads": [read_counts.sum()],
        "overall_umi_pct": [np.round(umi_counts.sum() / read_counts.sum() *
100, 2) if read_counts.sum() > 0 else np.nan]
    })
    result.to_csv(Path(args.output_csv), index=False)
    summary.to_csv(Path(args.output_csv).with_suffix(".summary.csv"),
index=False)
```

Appendix 3. demultiplex_bam.sh

Open a text editor and save the following code in a file called: `demultiplex_bam.sh`

```
#!/usr/bin/env bash
set -euo pipefail

if [[ $# -ne 4 ]]; then
    echo "Usage: $0 <output_dir> <bam_file> <barcode_file> <tag_name>" >&2
    exit 1
fi

output_dir="$1"
bam_file="$2"
barcode_file="$3"
tag_name="$4"

mkdir -p "$output_dir"

while IFS=$'\t' read -r sample_id tag_value extra || [[ -n "$sample_id" || -n
"$tag_value" || -n "$extra" ]]; do
    sample_id="${sample_id%$'\r'}"
    tag_value="${tag_value%$'\r'}"
    extra="${extra%$'\r'}"

    # Allow blank and comment lines in the barcode mapping file.
    if [[ -z "$sample_id" && -z "$tag_value" && -z "$extra" ]]; then
        continue
    fi

    if [[ "$sample_id" =~ ^[[:space:]]*# ]]; then
        continue
    fi

    picard FilterSamReads \
        I="${bam_file}" \
        O="${output_dir}/${sample_id}.bam" \
        TAG="${tag_name}" \
        TAG_VALUE="${tag_value}" \
        FILTER=includeTagValues
done < "${barcode_file}"
```

Ensure the script is executable by running the following command from the file's directory:

```
chmod a+x demultiplex_bam.sh
```

ALITHEA GENOMICS

info@alitheagenomics.com

www.alitheagenomics.com

Alithea Genomics SA
Batiment Serine
Rue de la Corniche 8, level -2
1066 Epalinges
Switzerland
Tel: +41 78 830 3139

Alithea Genomics Inc.
321 Ballenger Center Drive
Suite 125 C/O Alithea Genomics Inc.
Frederick, MD 21703
USA
Tel: +1 240 656 3142