Methylation Analysis: .fastq to Multi-omics Integration (Revision B)

Overview

This document presents the steps that are required to perform computational analysis of single-cell methylation data according to the EPI-Clone method.[1]

1. Generation of count matrices for methylation and antibody assays.
2. Detection of cell clones based on their methylation signatures.
3. Multiomics integration of multiple assays

Requirements

Software

Methylation analysis requires the following software tools.

1. Tapestri Pipeline with Experimental DNA + Protein pipeline enabled. To get access to Tapestri Pipeline, please contact Mission Bio support: support@missionbio.com
2. EPI-Clone pipeline. For more information, please refer to the EPI-clone GitHub repository.
3. Tapestri Mosaic package. For details, please see Mission Bio GitHub repository.

Files

Methylation analysis begins with the Tapestri Pipeline step, so most of the input files match the inputs of DNA + Protein run (details can be found in the Getting Started Guide).

1. Methylation sequencing files in .fastq.gz format. Methylation files match DNA sequencing files from the standard DNA + protein workflow.
2. Protein sequencing files in .fastq.gz format.
3. Methylation panel files. These files match DNA panel files from the standard DNA + protein workflow.
4. Protein panel files.

In addition, methylation analysis requires a pipeline configuration file in .yaml format, which needs to be prepared by the user. Cell calling for methylation analysis is not performed on all amplicons, but only on a subseti.e. on amplicons without a cut site for the methylation-sensitive enzyme). Amplicons without the enzyme cut site (“control” amplicons) need to be determined during the planning stage of the experiment (see Amplicon Panel Design Guidelines for details).

The `drop_amplicons` argument in the configuration file is used to specify all non-control amplicons that should not be used for cell calling (i.e., any amplicon with a target CpG cut site). Only the remaining control amplicons will be used for cell calling. Although non-control amplicons are dropped from cell calling, the remaining pipeline steps take all the amplicons into account so the final count matrix file contains counts per barcode for all the amplicons in the panel.

```
cellfinder:
   drop_amplicons:
   - AMPL376447
   - AMPL371688
   - AMPL371723
   - AMPL371137
   - AMPL371170
   - AMPL366134
   - AMPL366093
   - AMPL365920
```

Figure 1. First ten lines of the configuration file (.yaml format) used to run the Experimental Tapestri Pipeline DNA v3.6. The drop_amplicon argument instructs the pipeline not to include the listed amplicons in the cell calling step.

The .yaml file can be added to the Tapestri Pipeline instance by using the Add Files button, specifying File Type as Other, and then pointing to the Tapestri Custom Configuration entry of the drop-down list (Figure 2).

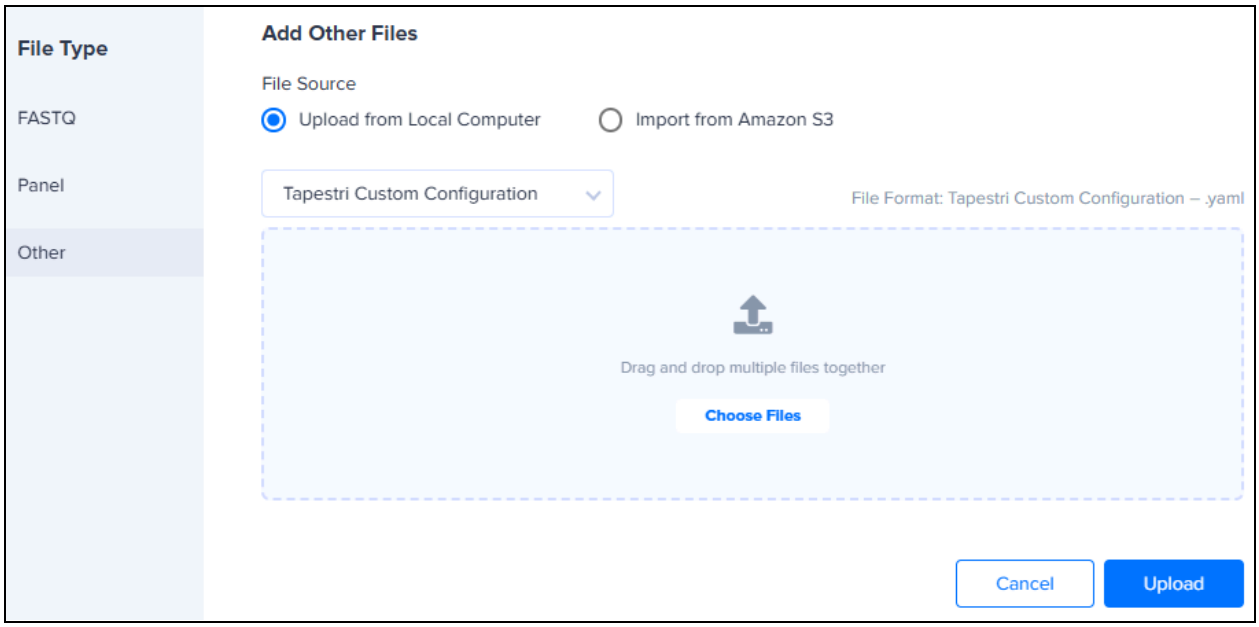

Figure 2. Adding configuration file (.yaml) to the Tapestri Pipeline.

Generation of Count Matrices

The first step of the methylation analysis is to obtain the count matrix file for each assay (i.e. methylation and antibody). This task is performed by Tapestri Pipeline. Getting Started Guide can be consulted for a general overview.

Importantly, methylation analysis is not based on the Standard DNA + Protein pipeline, but on the Experimental DNA + Protein pipeline. Experimental DNA + Protein pipeline can be selected from the Select the Pipeline drop-down menu, when launching the run (Figure 3).

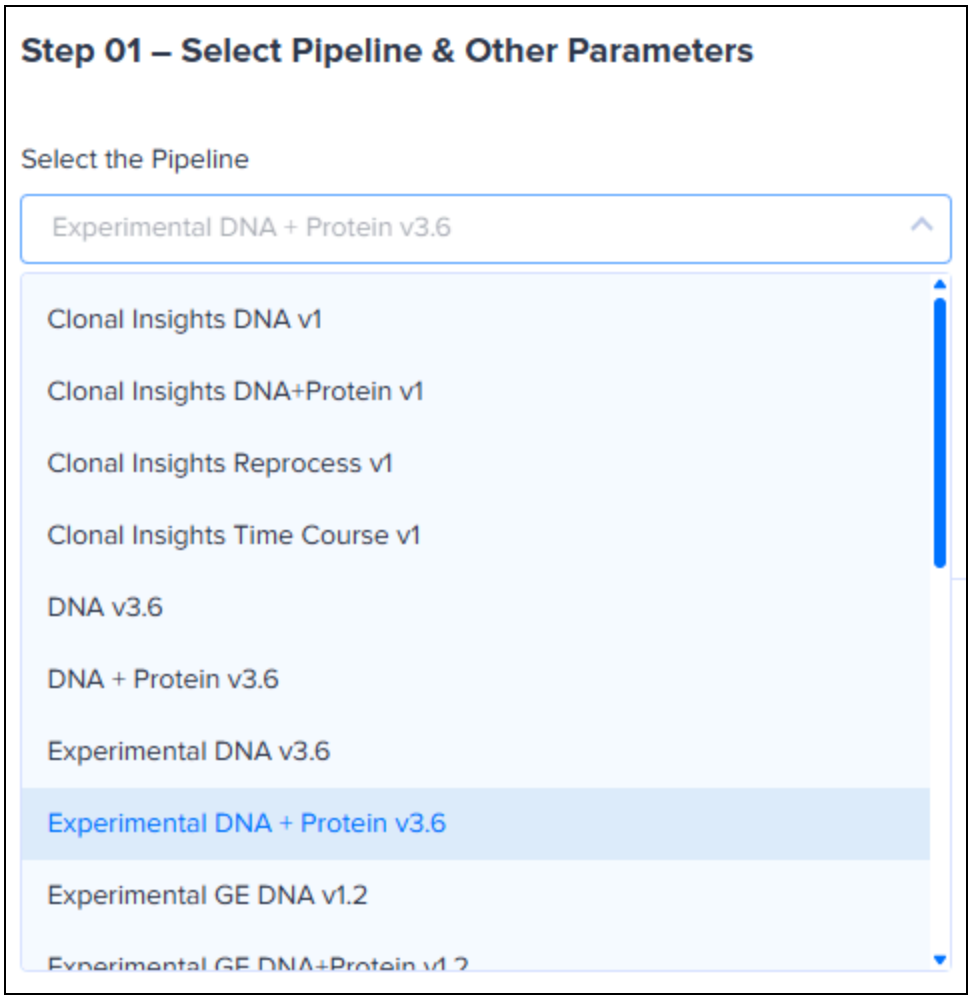

Figure 3. Specifying Experimental DNA + Protein pipeline when launching a new Tapestri Pipeline run.

Moreover, a mandatory step of the methylation analysis is to provide the configuration file with the list of amplicons that should *not* be used for cell calling (as described in the Requirements section). The file should be specified in the Select the Config drop-down menu when launching the run (Figure 4).

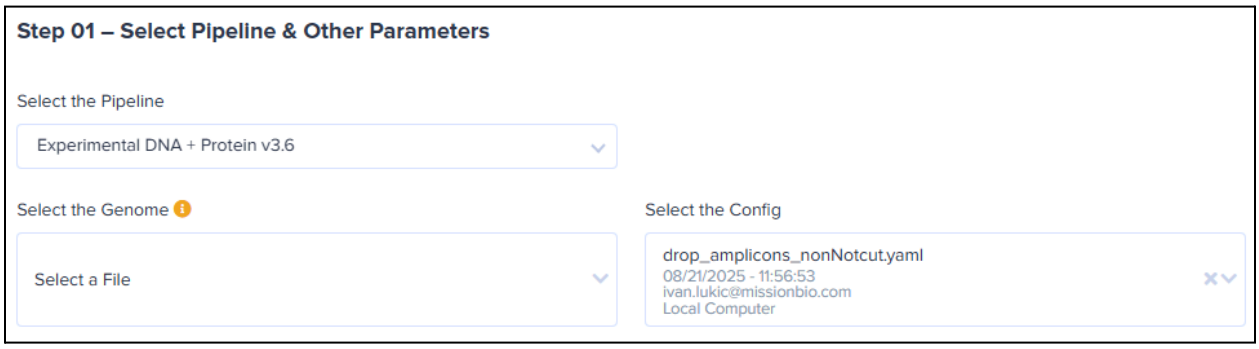

Figure 4. Selecting the config file when launching a new Tapestri Pipeline run.

Methylation Count Matrix

The methylation count matrix file is the same as the DNA count matrix file generated by the Standard DNA + Protein pipeline; it reports the number of forward reads assigned to each amplicon for each cell called in the run (more on the topic can be found in this document). The default suffix is *.cell.barcode.distribution.tsv.zip and the file can be downloaded from the Output Files tab of the run report. An example of the methylation count matrix is shown in Figure 5.

cell_barcode	AMPL355410		AMPL355412		AMPL355442		AMPL355513		AMPL355524	
AACAACCTACCTCGGTCA	0		0	0	0	0	180	11	10	0
AACAACCTACTTGGATCA	0		0	0	0	0	0	0	17	0
AACAACCTATCAAGCGTA	0		0	0	3	7	0	0	8	0
AACAACTGGGTTCAGAAG	4		9	0	0	7	33	0	0	5
AACAACTGGTGCCAGACG	0		0	0	0	21	34	0	0	0
AACAATGCAGTCAGCGTC	0		0	0	4	3	7	1	24	0
AACAATGCAGTTATCGAT	0		0	0	3	0	25	0	0	0
AACACACTCCGTGCAAGG	0		0	0	2	2	0	0	2	0
AACAGATCGAGAAGAGTC	0		0	0	0	0	0	0	0	0
AACAGATCGAGATCTGAC	0		0	0	20	42	10	1	30	0
AACAGATCGCATTATGTC	0		0	0	0	20	42	1	22	0
AACAGATCGCGCAACCTC	4		0	0	4	0	0	0	0	0
AACAGATCGCGTGCCTAT	0		0	0	0	0	14	0	0	7
AACAGATCGTGAGCAGAT	0		9	0	4	2	0	0	0	0

Figure 5. A snippet of the methylation count matrix file (barcodes × amplicons).

Antibody Count Matrix

A file that would be complimentary to the methylation count matrix file, but presenting protein data, is not reported by Tapestri Pipeline, so it needs to be obtained in one of the following two ways.

1. Customers working with on-prem instance of Tapestri Pipeline can look in the `*/protein/results/tsv` directory of the run and use the *-counts.tsv file. Note that the *-counts.tsv file is in the long format (Figure 6), so it has to be pivoted in the wide format.

```
cell_barcode    ab_barcode_description  umi_count
AACAACCTAAACAACCTA-1    Esam    1
AACAACCTAAACAACCTA-1    MHC_II    1
AACAACCTAAACAGCAGT-1    CD127    1
AACAACCTAAACCATGGT-1    CD127    1
AACAACCTAAACCATGGT-1    CD48    2
```

Figure 6. A snippet of the protein count matrix in the long format.

The following R code reshapes the protein count matrix from the long to the wide format. The code uses the same dependencies as listed in the Preparation Steps section (below).

```
None
# load the protein count matrix
AB_TSV    <- "/full/path/to/ab_counts.tsv"
dt_ab <- fread(AB_TSV)

# cast from long to wide
ab_wide <- dcast(
  dt_ab,
  cell_barcode ~ ab_barcode_description,
  value.var = "umi_count",
  fun.aggregate = sum,
  fill = 0
)

# (optional) sanity check
cat("AB (wide) dims (cells x proteins): ", paste(dim(ab_wide), collapse = " x
"), "\n", sep = "")

# example output of the sanity check
# AB (wide) dims (cells x proteins): 656825 x 21
```

2. An alternative approach is to parse the antibody count data from the standard .h5 file, which is available via the Output Files tab of the run report (file suffix is *.dna+protein.h5 and the description is here).

Add the snippet below to the DNA + Protein jupyter notebook to generate the .tsv file that is needed for downstream analysis.

```python
# load the data
sample = ms.load(h5path, raw=False, filter_variants=True, single=True,
whitelist = [], filter_cells=False)
# create a data frame with antibocy counts
reads = sample.protein.get_attribute('read_counts', constraint='row+col')
# edit the index header
reads.index.names = ['cell_barcode']
# save the matrix as a .tsv file
reads.to_csv(file_name, sep='\t')
```

An example of an antibody count matrix file is shown in Figure 7.

cell_barcode	CD11b	CD19	CD3	CD33	CD34	CD38	CD45	CD90	HLA-DR	Mouse_IgG1k
AACAACCTACTCTAGTAA	2	0	0	10	2	2	3	6	7	0
AACAACCTATCCTTAGGT	7	2095	16	14	103	2189	2428	4	13171	16
AACAACTGGAATCAGGAG	1	116	0	3	2	244	59	0	803	0
AACAACTGGAATGTGCGT	14	3469	33	57	261	7964	2260	25	46627	14
AACAACTGGATCGAATCG	1	0	0	0	4	1	2	3	9	0
AACAACTGGTAGACCATG	3	3	8	6	4	4	20	4	52	0
AACACACTCCACTGGTCC	0	47	0	9	6	62	10	0	120	0
AACACACTCTCAAGCGTA	0	5	0	39	304	241	70	1	103	1
AACAGATCGCTATCCATC	0	97	1	0	5	390	41	0	177	0
AACCATGGTCACGGATAG	0	29	0	0	5	141	22	0	88	0

Figure 7. A snippet of the antibody count matrix file in a .tsv format. The file (wide format) was prepared using the python block provided above.

Cell Clustering

The second step of the methylation analysis is to delineate cell clones based on their methylation signatures. Cell clustering starts with count matrix files created in the previous step and assigns each cell to a clone. The output of cell clustering is a file that lists each cell barcode and its corresponding cluster assignment.

A prerequisite for clustering is to install the EPI-clone pipeline. Note that EPI-clone is written in R (unlike Tapestri Mosaic, which is a python package).

Preparation Steps

Point to the location of the methylation and antibody count matrices.

```
None
# provide paths to .tsv files
DNAM_TSV  <- "/full/path/to/dnam_counts.tsv"
AB_TSV    <- "/full/path/to/ab_counts.tsv"
```

Load packages (previously installed).

```
None
# provide paths on your machine
HELPERS_R  <- "~/EPI-clone/scripts/helper_functions.R"
EPICLONE_R <- "~/EPI-clone/scripts/EPIClone.R"

suppressPackageStartupMessages({
  library(data.table)
  library(Matrix)
  library(Seurat)
  library(SeuratObject)
  library(ggplot2)
})

source(HELPERS_R)   # defines plot_type_methylation()
source(EPICLONE_R)  # defines epiclone()
```

Loading HELPERS_R may result in a warning message (below). The message is harmless and may be ignored.

```
Warning message:
The `size` argument of `element_line()` is deprecated as of ggplot2 3.4.0.
Please use the `linewidth` argument instead.
This warning is displayed once every 8 hours.
Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```

Loading EPICLONE_R prints a number of messages. Successful installation of the CHOIR package is confirmed by the message below.

Prepare output directories.

```
None
dir.create("plots",   showWarnings = FALSE, recursive = TRUE)
dir.create("exports", showWarnings = FALSE, recursive = TRUE)
```

Set seed for the random number generator (optional).

```
None
set.seed(42) # suggested by Deep Thought, feel free to modify
```

Loading and Preparing the Matrix Files

Load the methylation matrix file.

```
None
dt_dnam <- fread(DNAM_TSV)

# inspect the data (optional)
list(
  dims = dim(dt_dnam),
```

```
  names_1_10 = names(dt_dnam)[1:min(10, ncol(dt_dnam))],
  preview = dt_dnam[1:3, 1:min(6, ncol(dt_dnam))]
)
```

The methylation matrix generated by the upstream pipeline has barcodes on rows and amplicons on columns. It needs to be transposed for the purpose of the downstream analysis.

```
None
# convert to matrix and transpose to amplicons × cells
data.table::setnames(dt_dnam, 1, "cell_barcode")
barcodes_dnam <- dt_dnam[["cell_barcode"]]
m_dnam <- as.matrix(dt_dnam[, -1, with = FALSE])
storage.mode(m_dnam) <- "integer"
rownames(m_dnam) <- barcodes_dnam
mat_dnam <- t(m_dnam)   # rows = AMPL*, cols = cells

# quick check (optional)
dim(mat_dnam)
rownames(mat_dnam)[1:3]
colnames(mat_dnam)[1:3]
```

Example output is shown below.

```
> dim(mat_dnam)
[1]  663 5478
> rownames(mat_dnam)[1:3]
[1] "AMPL366556" "AMPL388182" "AMPL388183"
> colnames(mat_dnam)[1:3]
[1] "TGAGAATCCCTCCACGTC-1" "TGTGCCATGACTATCCTG-1"
"ACCAGCGTTCTAGTGCTG-1"
```

Next, load the antibody count matrix.

```
None
# read full AB table (wide format)
ab_wide <- fread(AB_TSV)
```

Subsetting Cells per Matrix File

Count the number of cells present in both matrices (methylation and protein) and compare.

```
None
# DNAm cells (from the mat_dnam)
cells_dnam <- colnames(mat_dnam)

# AB cells (from the wide table you just made)
cells_ab <- ab_wide$cell_barcode

# intersection (cells present in BOTH assays)
cells_common <- intersect(cells_dnam, cells_ab)

# quick peek (optional)
length(cells_dnam)    # how many DNAm cells total
length(unique(cells_ab))  # how many AB cells total
length(cells_common)  # how many cells overlap
```

Example output is below.

```
> length(cells_dnam)    # how many DNAm cells total
[1] 5478
> length(unique(cells_ab))  # how many AB cells total
[1] 656825
> length(cells_common)  # how many cells overlap
[1] 5470
```

Both matrices are subset, to keep only cells present in both assays (methylation and protein).

```
None
# subset DNAm to the common cells (keeps amplicons × cells)
mat_dnam_common <- mat_dnam[, cells_common, drop = FALSE]

# subset AB rows to just the common cells, preserving their order
# reorder ab_wide rows to match 'cells_common' exactly
idx <- match(cells_common, ab_wide$cell_barcode)
stopifnot(!any(is.na(idx))) # optional
```

```
ab_wide_sub <- ab_wide[idx, ]  # rows now exactly in cells_common order
stopifnot(identical(ab_wide_sub$cell_barcode, cells_common)) # optional

# convert to matrix using base subsetting (works for data.frame or data.table)
ab_cxg <- as.matrix(ab_wide_sub[, -1, drop = FALSE])  # drop the first
'cell_barcode' column
storage.mode(ab_cxg) <- "integer"
rownames(ab_cxg) <- ab_wide_sub$cell_barcode

# transpose to proteins × cells for the Seurat package
ab_mat <- t(ab_cxg)

# quick checks (optional)
dim(mat_dnam_common) # amplicons × cells (before DNAm QC)
dim(ab_mat) # proteins × cells
identical(colnames(mat_dnam_common), colnames(ab_mat))
```

Expected output is below. Both matrices should now have the same number of cells (i.e. 5470 in the current example) and their barcodes should match (TRUE).

```
> dim(mat_dnam_common)
[1]  663 5470
> dim(ab_mat)
[1]   20 5470
> identical(colnames(mat_dnam_common), colnames(ab_mat))
[1] TRUE
```

Coverage-based Filtering

Next, cells and amplicons in the methylation assay will be filtered for coverage. Ideally, amplicons with sufficient coverage are determined experimentally in the undigested control experiment: for example, select amplicons with at least 1 read in more than 90% of the cells.

If the experimental data are not available, you may want to perform exploratory analysis of the count matrix to determine the minimum number of reads per amplicon (across all the cells) and/or minimum total reads per cell. Values shown below may be too “aggressive” for some experiments, so they need to be optimised based on the actual experimental data. Any cells removed from the methylation assay need to be removed from the protein assay, too.

```
None
# coverage threshold
min_amp_reads  <- 50    # minimum total reads per amplicon (row)
min_cell_reads <- 500   # minimum total reads per cell (column)

n_amp_before <- nrow(mat_dnam_common); n_cell_before <- ncol(mat_dnam_common)

keep_amps  <- rowSums(mat_dnam_common) >= min_amp_reads
keep_cells <- colSums(mat_dnam_common) >= min_cell_reads

# apply to DNAm
mat_dnam_qc <- mat_dnam_common[keep_amps, keep_cells, drop = FALSE]

# apply the same CELL filter to AB to keep assays aligned
ab_mat_qc <- ab_mat[, colnames(mat_dnam_qc), drop = FALSE]

# quick check (optional)
cat("DNAm amplicons kept:", sum(keep_amps), "/", n_amp_before, "\n")
cat("DNAm cells kept    :", sum(keep_cells), "/", n_cell_before, "\n")
cat("Shapes after QC    : DNAm ", paste(dim(mat_dnam_qc), collapse = "x"),
    " | AB ", paste(dim(ab_mat_qc), collapse = "x"), "\n", sep = "")
```

Expected output is below.

```
> cat("DNAm amplicons kept:", sum(keep_amps), "/", n_amp_before, "\n")
DNAm amplicons kept: 662 / 663
> cat("DNAm cells kept    :", sum(keep_cells), "/", n_cell_before, "\n")
DNAm cells kept    : 5470 / 5470
> cat("Shapes after QC    : DNAm ", paste(dim(mat_dnam_qc), collapse = "x"),
+     " | AB ", paste(dim(ab_mat_qc), collapse = "x"), "\n", sep = "")
Shapes after QC    : DNAm 662x5470 | AB 20x5470
```

Binarising Methylation Data

The methylation matrix will be binarised (improves robustness of the analysis). Cut-off value was set to 1 (i.e. one or more reads per amplicon per cell is considered as “signal” / methylated). You may want to optimise based on your own data.

```
None
# load packages
library(Seurat); library(SeuratObject); library(Matrix)

# binarize DNAm (>=1 reads considered methylated)
min_reads_call <- 1
bin_mat <- (mat_dnam_qc >= min_reads_call) * 1L
bin_sp  <- Matrix(bin_mat, sparse = TRUE)
```

Preparation of the Seurat Object

The epiclone() function calls the Seurat package, so both matrices (methylation and protein) need to be added to the Seurat object as assays. epiclone() also requires some pieces of metadata, some of which need to be determined experimentally. In this vignette, such values were replaced by “placeholder” values. Be sure to replace it with the actual metadata.

Note: A new version of Seurat has been released between the development of EPI-Clone and the time of writing of this vignette. This vignette is based on Seurat v5.

```
None
# Seurat object with DNAm
seu <- CreateSeuratObject(counts = bin_sp, assay = "DNAm")
DefaultAssay(seu) <- "DNAm"

# add AB assay (proteins × cells)
ab_sp <- Matrix(ab_mat_qc, sparse = TRUE)
# command below will result in a harmless warning
seu[["AB"]] <- CreateAssayObject(counts = ab_sp)

# minimal metadata epiclone() expects
seu$Sample <- "sample_with_AB"
dn_counts <- SeuratObject::GetAssayData(seu, assay = "DNAm", layer = "counts")
# adding placeholder values; true values need to be determine experimentally
cell_tot  <- Matrix::colSums(dn_counts)
cutpoint  <- stats::median(cell_tot)
seu$seurat_clusters     <- factor(ifelse(cell_tot > cutpoint, "high", "low"))
seu$cleanClone          <- NA
seu$avgNNdist           <- 0
seu$PerformanceNonHhaI  <- 0

# Seurat v5 layers
```

```
seu <- SeuratObject::SetAssayData(seu, assay = "DNAm", layer = "data", new.data
= dn_counts)
ab_counts_layer <- SeuratObject::GetAssayData(seu, assay = "AB", layer =
"counts")
seu <- SeuratObject::SetAssayData(seu, assay = "AB", layer = "data", new.data =
ab_counts_layer)

# quick check (optional)
SeuratObject::Layers(seu[["DNAm"]])
SeuratObject::Layers(seu[["AB"]])
dim(seu)
```

Example output is shown below.

```
> SeuratObject::Layers(seu[["DNAm"]])
[1] "counts" "data"
> SeuratObject::Layers(seu[["AB"]])
[1] "counts" "data"
> dim(seu)
[1]  662 5470
```

Filtering Amplicons on Methylation Variability

Another step of filtering prepares amplicons for clustering. The idea of clustering is to delineate cell clones based on their methylation profiles, so we are keeping only amplicons that show sufficient variability across all the cells. There are two ways to do it:

a) Select amplicons based on a custom approach.
b) Use built-in EPI-Clone functionality to detect static CpGs.

Manual Selection of Amplicons

A possible filtering strategy that can be used as a custom approach is to select differentially methylated amplicons based on a threshold, such as methylated in between 25% and 90% of the cells (shown below). The exact values should be optimised based on your own data.

```
None
mr <- Matrix::rowMeans(SeuratObject::GetAssayData(seu, assay = "DNAm", layer =
"counts"))
selected <- names(mr)[mr >= 0.25 & mr <= 0.90]

# quick check (optional)
length(selected); summary(mr[selected])
```

Example output is shown below.

```
> length(selected); summary(mr[selected])
[1] 417
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.2516  0.4804  0.6771  0.6352  0.7958  0.8991
```

Running EPI-Clone

For automatic selection of variable amplicons, set the EPI-Clone's option `selected.CpGs` to `NULL`. Please note that EPI-Clone's selection of static CpGs works better for aggregated data sets (aggregation of multiple samples) and may not be ideal for analysis of a single sample.

The epiclone() function is used to detect methylation-based clones. Ground-truth labels need to be determined experimentally (e.g. LARRY barcodes), but are not essential to perform methylation-based clustering.

```
None
# make sure the functions are available (optional)
if (!exists("epiclone")) source("~/EPI-clone/scripts/EPIClone.R")
if (!exists("plot_type_methylation"))
source("~/EPI-clone/scripts/helper_functions.R")

# create directory to store plots in
```

```
dir.create("plots", showWarnings = FALSE)

res <- epiclone(
  seurat_obj              = seu,
  selected.CpGs           = selected, # set to NULL to use EPI-Clone
  plotFolder              = "plots",
  tuneParams              = FALSE,
  runCHOIR                = FALSE,
  trueClone               = NULL, # no ground-truth labels
  methylation.assay.name  = "DNAm",
  protein.assay.name      = "AB"
)
```

Image below shows a successful epiclone() run. Warnings are harmless.

```
Identifying big clones...

|======================================================================|
100%
Warning: The default method for RunUMAP has changed from calling Python UMAP via
reticulate to the R-native UWOT using the cosine metric
To use Python UMAP via reticulate, set umap.method to 'umap-learn' and metric to
'correlation'
This message will be shown once per session
Using method 'umap'
0%   10   20   30   40   50   60   70   80   90   100%
[----|----|----|----|----|----|----|----|----|----|
**************************************************|
Completed selection of big clones
Now clustering clones...

|======================================================================|
100%
Using method 'umap'
0%   10   20   30   40   50   60   70   80   90   100%
[----|----|----|----|----|----|----|----|----|----|
**************************************************|
Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck

Number of nodes: 4779
Number of edges: 56792

Running Louvain algorithm...
0%   10   20   30   40   50   60   70   80   90   100%
```

```
[----|----|----|----|----|----|----|----|----|----|
**************************************************|
Maximum modularity in 10 random starts: 0.4352
Number of communities: 49
Elapsed time: 0 seconds
Warning messages:
1: The `slot` argument of `FetchData()` is deprecated as of SeuratObject 5.0.0.
Please use the `layer` argument instead.
The deprecated feature was likely used in the Seurat package.
  Please report the issue at <https://github.com/satijalab/seurat/issues>.
This warning is displayed once every 8 hours.
Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
2: `aes_string()` was deprecated in ggplot2 3.0.0.
Please use tidy evaluation idioms with `aes()`.
See also `vignette("ggplot2-in-packages")` for more information.
The deprecated feature was likely used in the Seurat package.
  Please report the issue at <https://github.com/satijalab/seurat/issues>.
This warning is displayed once every 8 hours.
Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```

Visualisation by UMAP

Finally, methylation-based clusters are visualised using UMAP. In the first step of the code block below, the Seurat object's metadata are examined to find the column with cluster IDs (in this example: clone_nn_res.5).

```
None
# Seurat result is object 'res'
# res contains another object 'obj'
# 'obj' contains the low dimensional representation
obj <- res$finalSeurat

# inspect column names to look for the one with the final clone assignments
> colnames(obj@meta.data)
```

Example output is shown below.

```
> colnames(obj@meta.data)
 [1] "orig.ident"         "nCount_DNAm"        "nFeature_DNAm"
 [4] "nCount_AB"          "nFeature_AB"         "Sample"
 [7] "seurat_clusters"    "cleanClone"         "avgNNdist"
```

```
[10] "PerformanceNonHhaI" "avgNNres"          "selected"
[13] "clone_nn_res.5"
```

If it is now obvious which column contains clone assignments, take a look at the one containing ‘clone’ in their label.

```
None
# inspecting clone_nn_res.5
obj$clone_nn_res.5
```

Partial output is below. The number of levels (49 in this example) matches the number of communities (i.e. clusters) detected by EPI-Clone (please compare with the EPI-Clone output above).

```
AATGCCTCAACTCGTCTA-1 ACCGGAGTACACCAAGAT-1 TGAGATCATGTGTCCGTG-1
           4              4              0
GGAGCATCTCGGTTCTAT-1 TGTACACGATAGACCGCA-1 TCCGGAGCTCTCTCGATA-1
           2              0              7
TAGCTGTTGACCATGTCT-1
          18
49 Levels: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 ... 48
```

After that, the column with clone labels is identified and the UMAP plot is invoked on low dimensional representation as generated by EPI-Clone (called `cloneUMAP`).

```
None
# obtaining the low dimensional representation as generated by EPI-Clone
obj$clone_id <- as.factor(obj$clone_nn_res.5) # use the column identified above
Idents(obj) <- obj$clone_id
DimPlot(obj, reduction = "cloneUMAP", group.by = "clone_nn_res.5", label =
TRUE)

# saving the plot as .png file
ggsave("plots/cloneUMAP_by_clone_nn_res5.png",
       plot = DimPlot(obj, reduction = "cloneUMAP",
                      group.by = "clone_nn_res.5", label = TRUE),
       width = 7, height = 6, dpi = 300)
```

An example of the resulting figure is shown below.

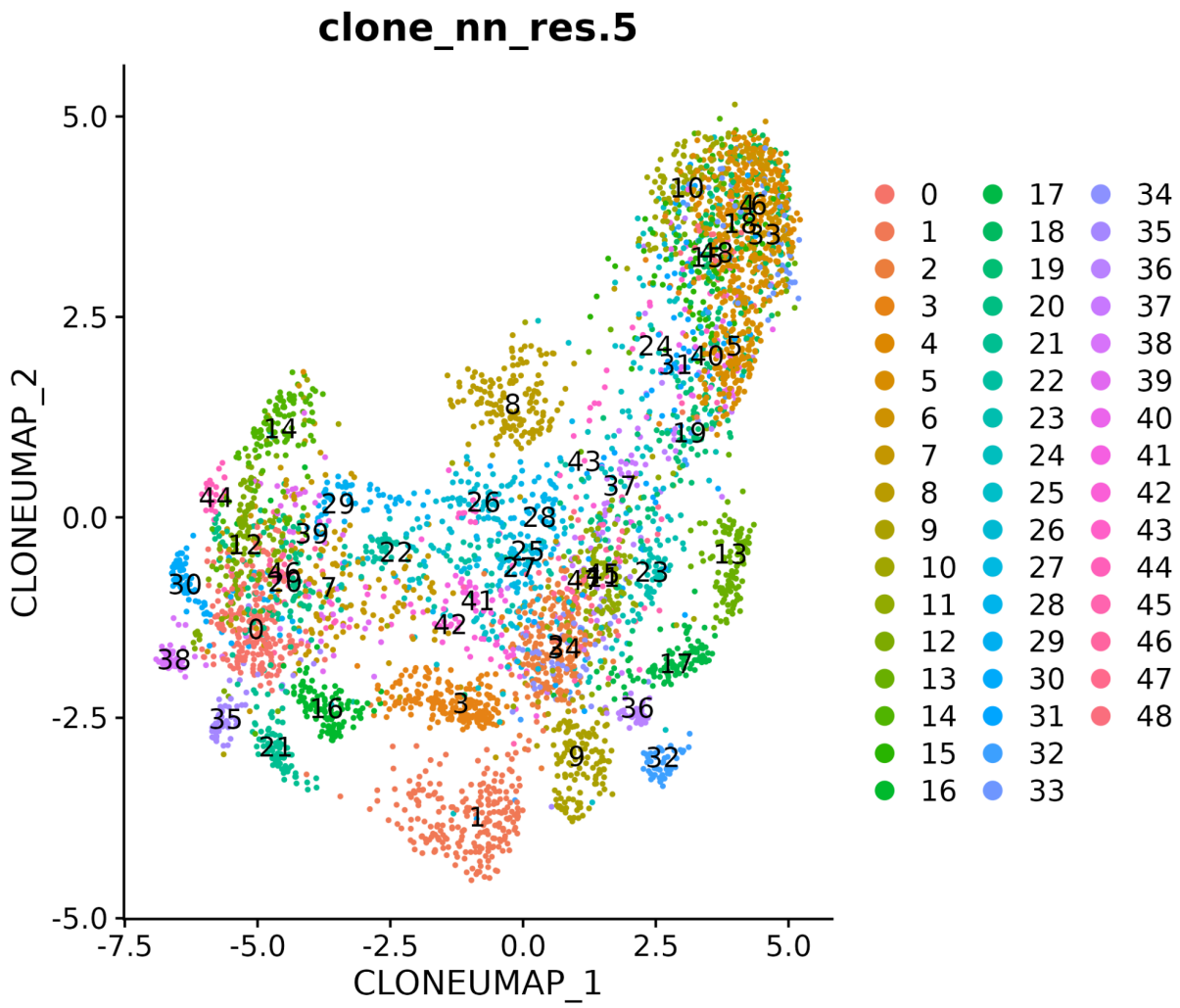

Figure 8. UMAP visualisation of methylation-based cell clusters. Each dot is a cell, colors show clusters.

Multiomics Integration

The third step of the analysis is multiomics integration, which integrates different assays. For instance, cell types of methylation clones detected in the previous step can be determined by overlaying protein staining data (antibody assay) on top of the methylation clusters.

Obtaining Cluster Assignments

At the clustering step, information on cluster membership of each barcode is contained within the Seurat object. The code below shows how to extract that information and save as a .tsv file with two columns.

```
None
# 1) pick the predicted-clone column (e.g., "clone_nn_res.5")
meta <- res$finalSeurat@meta.data
clone_cols <- setdiff(
  colnames(meta)[startsWith(colnames(meta), "clone_") | grepl("clone_nn",
colnames(meta), fixed = TRUE)],
  "cleanClone"
)
stopifnot(length(clone_cols) >= 1)
clone_col <- clone_cols[1]   # or set explicitly: clone_col <- "clone_nn_res.5"

# 2) build the two-column data frame
out <- data.frame(
  cell_barcode        = rownames(meta),
  methylation_cluster = meta[[clone_col]],
  check.names = FALSE
)

# 3) write .tsv file to the current directory
write.table(out, file = "clone_assignments.tsv", sep = "\t", quote = FALSE,
row.names = FALSE)

# (optional) quick checks
head(out, 3)
table(out$methylation_cluster)[1:10]
file.exists("clone_assignments.tsv")
```

Example output is shown below. clone_assignments.tsv file has two columns:

1. cell_barcode
2. methylation_cluster: cluster ID (shown as colours in Figure 8)

```
> head(out, 3)
       cell_barcode methylation_cluster
```

```
1 TGAGAATCCCTCCACGTC-1                21
2 TGTGCCATGACTATCCTG-1                10
3 ACCAGCGTTCTAGTGCTG-1                10
> table(out$methylation_cluster)[1:10]

  0   1  10  11  12  13  14  15  16  17
232 225 144 144 144 140 139 132 126 120
> file.exists("clone_assignments.tsv")
[1] TRUE
```

Working with Cluster Assignments in Tapestri Mosaic

Once the cluster assignments per barcode is obtained (e.g. stored in *clone_assignments.tsv file*) they can be combined with other Tapestri assays using Tapestri Mosaic. For example, a protein-based UMAP can be colored by methylation-based clones to identify the cell types. Alternatively, DNA data can be combined with methylation-based clones to identify the existence of mutations within cells with particular methylation signatures.

The analysis will be illustrated based on the second example above, i.e. joining genotyping information with methylation results. Briefly, DNA data were processed as follows: the .h5 file has been loaded to a Tapestri Mosaic notebook, two variants were selected, and three clones were defined based on zygosities of those variants. After that, PCA and UMAP were invoked on the VAF data for those variants. The image below shows cell clustering based on VAFs of two variants.

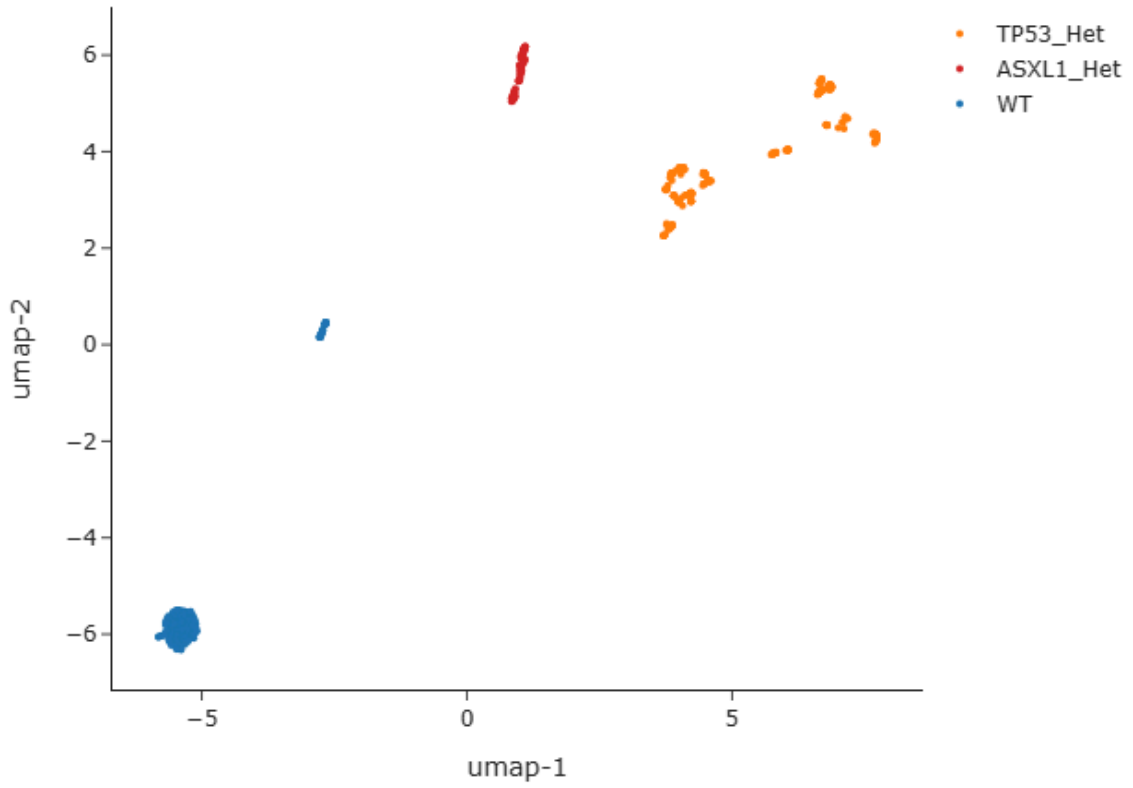

Figure 9. UMAP visualization of cell clustering based on VAFs of two variants detected in the sample. Each dot is a single cell. Colors indicate mutation-defined clones (i.e. TP53_Het, ASXL1_Het, and WT).

The following code snippets may be added to one of the standard Tapestri Mosaic notebooks. Note that, depending on the data and the goal of the analysis, the steps and commands below may need to be modified.

The first code block loads dependencies and the clone_assignments.tsv file (generated using the *Cell Clustering* step of this guide.

```python
# loading packages
import os
import pandas as pd

# loading clone_assignments.tsv file
# if needed change the working directory using the os package
# import os
```

```
# os.chdir("/path/to/the/directory/with/tsv/file")
clone_assignments = pd.read_csv("clone_assignments.tsv", sep = '\t')
```

Next, barcodes shared by both the DNA assay and methylation assay need to be identified.

Python
```
# creating a list of barcodes shared by both assays
methylation_barcodes = clone_assignments['cell_barcode'].to_list()
dna_barcodes = sample.dna.row_attrs['barcode'].tolist()
common_barcodes = np.intersect1d(dna_barcodes, methylation_barcodes).tolist()
```

Both data objects are filtered to keep the barcodes present in both assays (i.e. those with available methylation information).

Python
```
# subset the sample.dna object to keep only barcodes present in both assays
sample.dna = sample.dna[common_barcodes, :]
# subset the clone_assignments data frame to keep oonly barcodes present in
both assays
clone_assignments_shared =
clone_assignments[clone_assignments['cell_barcode'].isin(common_barcodes)]
```

The code below shows how to sort the *clone_assignments_shared* data frame so that the order of the barcodes matches the one in the *sample.dna* object. Once that is done, the clonal assignment of each barcode will be stored as a list (*methylation_clusters*).

Python
```
# get the barcodes present in the filtered sample.dna object
dna_barcodes_shared = sample.dna.row_attrs['barcode'].tolist()

# sort the cell_barcode column of the clone_assignmets_shared data frame to
match the order of the barcodes in the dna_barcodes_shared list
clone_assignments_shared = clone_assignments_shared.copy()
```

```
clone_assignments_shared['cell_barcode'] =
pd.Categorical(clone_assignments_shared['cell_barcode'],
categories=dna_barcodes_shared, ordered=True)

clone_assignments_sorted = clone_assignments_shared.sort_values('cell_barcode')

# extract the properly sorted list of clone assignments of each barcode
methylation_clusters = clone_assignments_sorted['methylation_cluster'].tolist()
```

Transfer of the methylation clusters to the sample.dna object is performed by replacing pre-existing labels of each barcode (row attribute ‘label’), with the help of the *_Assay.set_labels()* function of Tapestri Mosaic.

Python

```
# optional: check the labels before the transfer
# sample.dna.row_attrs['label']

# transfer methylation clusters to the sample.dna object
sample.dna.set_labels(methylation_clusters)

# optional: check the labels before the transfer
# sample.dna.row_attrs['label']
```

Finally, Figure 10 is a re-creation of Figure 9, but with colors indicating methylation-based clusters.

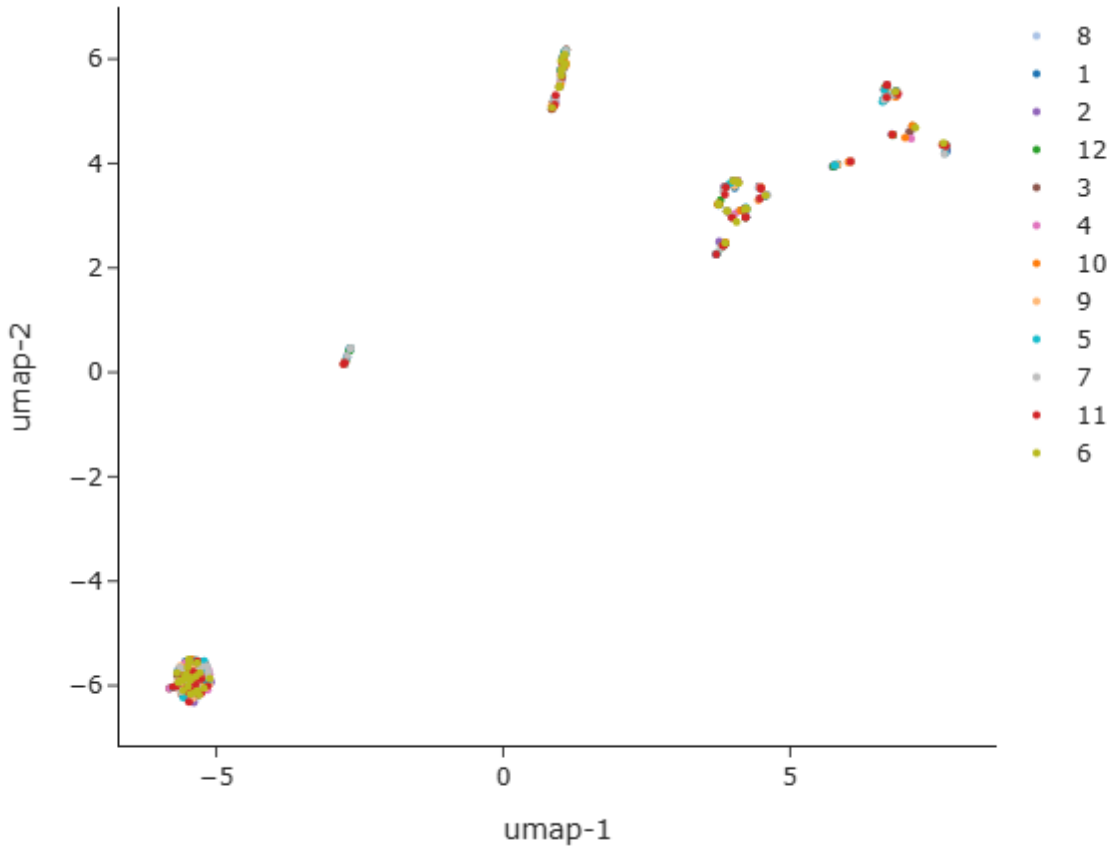

Figure 10. UMAP visualisation of cell clustering based on VAFs of two variants detected in the sample. Each dot is a single cell. Colors indicate methylation-defined clones.

References

1. Scherer, M., Singh, I., Braun, M.M. et al. Clonal tracing with somatic epimutations reveals dynamics of blood ageing. Nature 643, 478–487 (2025). https://doi.org/10.1038/s41586-025-09041-8
2. Bianchi, A., Scherer, M., Zaurin, R. et al. scTAM-seq enables targeted high-confidence analysis of DNA methylation in single cells. Genome Biol 23, 229 (2022). https://doi.org/10.1186/s13059-022-02796-7