Fluent BioSciences

PIPseeker™ v3.3

User Guide

Doc ID: **FB0002787**
Revision: **8**
State: **Release**

Table of Contents

Introduction

PIPseeker™ analyzes single-cell data obtained with Fluent BioSciences' proprietary PIPseq™ V 3' Single Cell RNA (scRNA-seq) Kits. PIPseeker offers a comprehensive analysis solution that provides the user with detailed metrics, gene expression profiles, basic cell quality and clustering indicators, and cell type annotation for some sample types. The outputs of PIPseeker can then be used for subsequent, specialized tertiary analysis streams. PIPseeker also supports specialized applications like synthetic nucleotide tags (SNTs) for measuring surface protein levels or CRISPR guide expression, and cell hashing using hashtag oligonucleotides (HTOs).

Pipeline Overview

This section describes the core PIPseeker workflow involving gene expression data from mRNA capture and/or data from synthetic nucleotide tags (SNT) . Depending on the nature of the experiment, all or some of the following steps will be performed:

- RNA (gene expression) data processing:
    1) FASTQ processing
    2) Mapping with *STAR* aligner
    3) Transcript counting
    4) Cell calling
    5) Clustering
    6) Differential expression
    7) Cell type annotation (optional)
- SNT data processing:
    1) FASTQ processing, including barcoding, MI error correction and SNT tag matching
    2) Extracting SNT metrics
    3) If RNA data is present, creating merged RNA+SNT raw and filtered count matrices
    4) If RNA data is not present, cell calling and clustering using SNT expression
    5) Creating feature plots for each SNT using the clustering results
- HTO data processing::
    1) FASTQ processing, including barcoding, MI error correction and HTO tag matching
    2) Extracting HTO metrics
    3) Creating merged RNA+HTO, RNA+SNT+HTO, or SNT+HTO raw and filtered count matrices
    4) Demultiplexing HTO data (i.e., identify singlets, multiplets, negative cells)

5) Creating HTO feature plots and diagnostic plots
6) Clustering for each demultiplexed sample
7) Creating SNT feature plots for each demultiplexed sample (when applicable)

- Generating an HTML summary report

FASTQ Processing

The purpose of this step is to:

- Assign barcodes to reads
- Remove unwanted technical sequences from the data before mapping
- Assign molecular identifiers (MIs) to reads
- Downsample the data (optional)

Structure of the FASTQ input files for PIPseq v4

PIPseq libraries are sequenced in a paired-end fashion, meaning the sequencing results will include separate FASTQ files for read 1 (R1) and read 2 (R2). Each read in R1 includes a barcode sequence, which identifies the individual PIP, followed by a molecular index sequence, identifying the specific mRNA molecule captured on each moiety. R2 includes the sequenced cDNA constructs created from the captured mRNA.

Structure of the FASTQ input files for PIPseq V

Fig. 1 describes the fragmentation process taking place in PIPseq V chemistry. Each read in R1 includes a barcode sequence, which identifies the individual PIP, followed by a 3-base binning index sequence. R2 includes the sequenced cDNA constructs created from the captured mRNA, which contain random cut sites that serve as intrinsic molecular identifiers (IMIs), and are used for molecular counting. The first 12 bases of R2 are used as the IMI for each read.

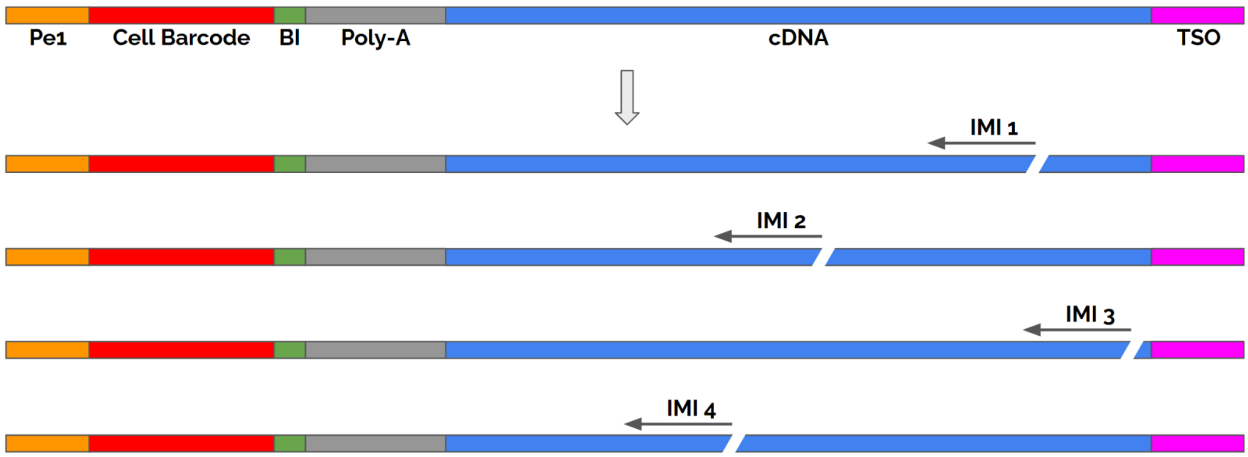

Figure 1: PIPseq V chemistry. Random enzymatic fragmentation sites create intrinsic molecular identifiers (IMIs). Up to fifteen different IMIs can be created from a single captured molecule using five cycles of whole transcriptome amplification. PE1 = Illumina Read 1 sequencing primer; BI = Binning Index.

Barcode Identification

The PIPseq platform uses a tiered barcode system, where every tier includes one of a specified list of possible barcodes. Barcode matching can therefore be done highly efficiently by matching each tier in isolation. PIPseeker allows a Hamming distance of 1 per tier, i.e., the bases in the R1 FASTQ corresponding to each tier's position can differ from a barcode by one base and still be matched to that barcode.

Before mapping with *STAR*, PIPseeker identifies the barcode combination and the intrinsic molecular identifier for each read. The tiered barcodes are replaced by a new set of generated barcodes. The generated barcodes and the IMIs are stored as part of the read header in a temporary, intermediate FASTQ file that is used by *STAR* for mapping. Intermediate FASTQ files are deleted at the end of the analysis.

R2 Trimming

The R2 file, which includes the cDNA construct, also includes technical sequences related to the PIPseq chemistry, namely the template switch oligo (TSO) and poly-A sequences. The prevalence of those sequences is inversely correlated with the fragment size. Because a read is less likely to be mapped accurately if it contains extraneous sequences, TSO sequences are removed from the 5' end and poly-A sequences are removed from the 3' end in the R2 FASTQ file. In addition, the first base, a constant T, is removed from the 5' end of R2. Reads that do not meet the minimum length criterion (20 bases) after these steps are discarded from further analysis.

Downsampling

In some experiments it may be desirable to standardize sequencing depth, so that experimental conditions could be reliably compared. PIPseeker allows downsampling of data to a specified number of reads using the `--downsample-to` flag. Reads are selected randomly, with the probability of selection determined by the total number of reads (provided using the `--input-reads` argument or determined automatically by counting the lines in the FASTQ inputs).

Outputs

The FASTQ processing step results in the following outputs:

- `<output path>/metrics/barcode_stats.csv`: General statistics pertaining to the quality of barcode matching.
- `<output path>/metrics/barcodes/barcode_whitelist.txt`: A list of all the generated barcodes (barcodes converted from the PIPseq 4-tier structure to a standard 16-base sequence)
- `<output path>/metrics/barcodes/generated_barcode_read_info_table.csv`: Detailed statistics for all the generated barcodes.

- `<output path>/barcoded_fastqs/*`: Intermediate FASTQ file(s) to be used by *STAR* for mapping. Those files include only the subset of reads that met the length requirement and have matching barcodes after the processing steps listed above. Those files include (in the read headers) the generated barcode for each read (index in the whitelist), followed by the molecular index, and the original (possibly trimmed) read from the R2 FASTQ file. When using multiple threads, a separate FASTQ file will be generated by each thread. Unless instructed otherwise using the `--retain-barcoded-fastqs` flag, PIPseeker deletes those files at the end of the analysis. Note that the original, raw FASTQ files are never deleted.

Mapping

PIPseeker uses *STAR* (Spliced Transcripts Alignment to a Reference) to map reads to a genome. *STAR* requires a special index to be built from the genome of interest and a corresponding transcript annotation. Some common indices, such as human and mouse, can be downloaded from the Fluent BioSciences website. However, for more unique situations, you may need to create your own *STAR* index using PIPseeker buildmapref.

Alignment Classification

PIPseeker can distinguish between reads mapping to exonic vs. intronic portions of a gene (Fig. 2). Intronic reads are derived from unspliced or "nascent" pre-RNAs, which are especially enriched in the nucleus. Because pre-RNAs are the direct precursors to fully-spliced RNAs, they are counted along with exonic transcripts by default. If the `--exons-only` flag is specified, PIPseeker will separate exonic from intronic mapping, and consider only exonic reads in subsequent analysis steps. Reads that map to intronic regions or beyond gene boundaries are discarded in this mode.

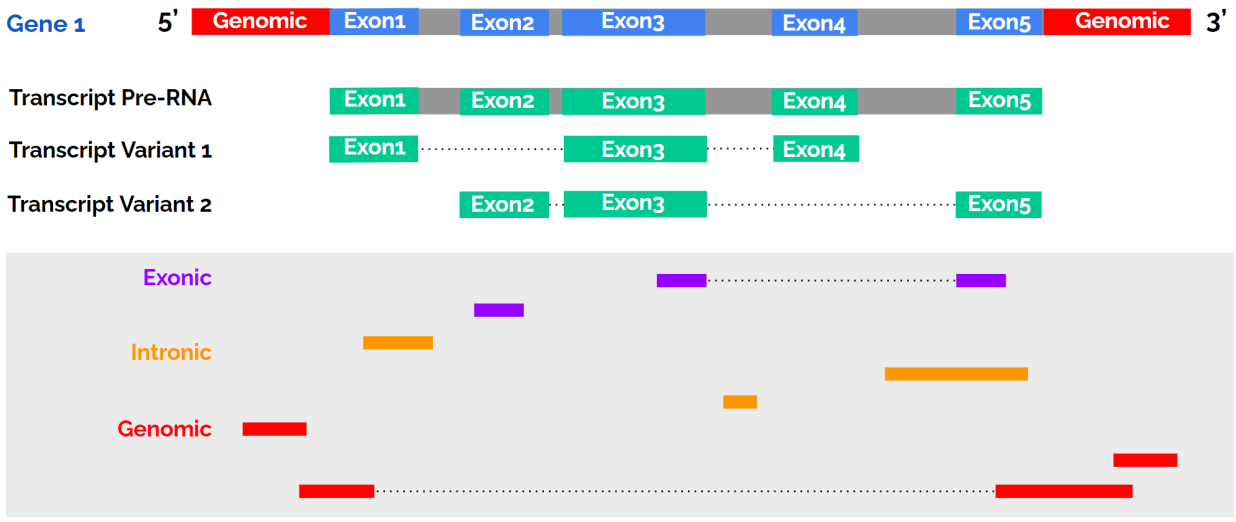

Figure 2: Alignment classification overview. An example gene (Gene 1) is shown with its genomic (red), exonic (blue), and intronic (gray) components. In this simple example, the unspliced transcription product (Transcript Pre-RNA) consists of all exons (1-5) and introns for the gene without splicing. Two transcript variants are shown, with spliced introns indicated as dashed lines. Example alignments are shown for **Exonic**: i) spanning multiple exons or ii) localized to a single exon, **Intronic**: i) starting in an exon but ending in an intron, ii) localized exclusively to an intron, and iii) starting in an intron and ending in an exon, **Genomic**: Any alignment to non-gene, unannotated segments of the reference genome.

Sorted BAM for downstream applications

The BAM file produced by *STAR* during the PIPseeker analysis is not sorted by genome position. Some users may want to have a sorted BAM compatible with various downstream applications, such as velocyto and scvelo. Running PIPseeker barcode with the `--sorted-bam` flag will generate a sorted BAM file (not used by PIPseeker), that can serve as an input to third party applications. The barcode and MI information will be incorporated as tags in the BAM file. Note that in PIPseq V, the MI will be a concatenation of the binning index and the IMI for each read. Using it in place of a UMI in the above applications may lead to inaccurate results for certain cell types.

Transcript Counting

PIPseeker constructs a raw count matrix, commonly referred to as a feature-barcode matrix. The count matrix output is a sparse representation of a matrix containing a full count of unique molecules for each gene and barcode.

Parsing the Mapping File

STAR stores the alignment information in a Binary Alignment Map (BAM) file, which provides information about all the reads in the input FASTQ files that were mapped to the genome. PIPseeker then parses the BAM file to create a data structure with all the molecules in the sample that were mapped to the reference. The molecule information is saved to an HDF5 format file called molecule_info.h5, and is used by PIPseeker to create the raw matrix.

MIs are corrected for sequencing errors using a Hamming distance tolerance of 1. Within each barcode, MIs are sorted by frequency of occurrence. If two IMIs are one base apart, the less common of them is converted to the more common IMI.

Molecular Counting for PIPseq v4

After the molecule information is gathered, PIPseeker assigns reads to genes by matching their alignment coordinates against the *STAR* mapping reference. If the `--exons-only` flag is used, gene assignments are restricted to exonic regions. Reads are then deduplicated by collapsing all reads with the same MI into a single count. The counts are written into a sparse matrix, where each element corresponds to a cell barcode and a gene.

Molecular Counting for PIPseq V

V chemistry generates fragments with unique starting positions, which serve as intrinsic identifiers that can be used for molecular counting (Fig. 3). Reads are grouped together based on the cell barcode and the assigned gene. Within each barcode and gene combination, IMIs are grouped in one of 64 bins, based on the 3-base binning index. For each bin, all identical IMIs are collapsed into a single count, since they are likely PCR duplicates of the same fragment generated during library prep. PIPseeker then counts the number of IMIs associated with each binning index and divides that number by a correction factor estimated from the data, which accounts for the additional copies generated from a single captured molecule during five amplification cycles. Finally, the corrected counts in each bin are summed to produce the final count estimate for this barcode and gene.

Estimating the correction factor is based on the fact that in some cases, the exact number of source molecules can be determined from the observed data for one barcode+gene combination. Examples of such cases include:

- 1 IMI in 1 bin ⇒ 1 molecule
- 2 IMIs in 2 bins ⇒ 2 molecule
- 3 IMIs in 3 bins ⇒ 3 molecules

We refer to these molecules as "uninflated", since each of those molecules has a single associated IMI. Other situation are ambiguous and include both uninflated molecules and "inflated" molecules, or molecules with multiple IMIs generated by fragmentation:

- 2 IMIs in 1 bin ⇒ 2 fragments from the same molecule or 2 different molecules
- 3 IMIs in 2 bins ⇒ 2 or 3 molecules

PIPseeker estimates the probability of one source molecule producing any number of copies between 1 and 15 from the portion of the data that has up to 15 IMIs in the same barcode+gene. It uses the cases with a known number of molecules to subtract the estimated uninflated portion from the cases with an unknown number of molecules, which results in a count of inflated molecules. The end result of this process is 15 probability values for generating up to 15 copies from a single source molecule. The correction factor is selected so as to minimize the number of inflated counts by setting a limit on the possible percentage of inflated counts to 1% of total molecules.

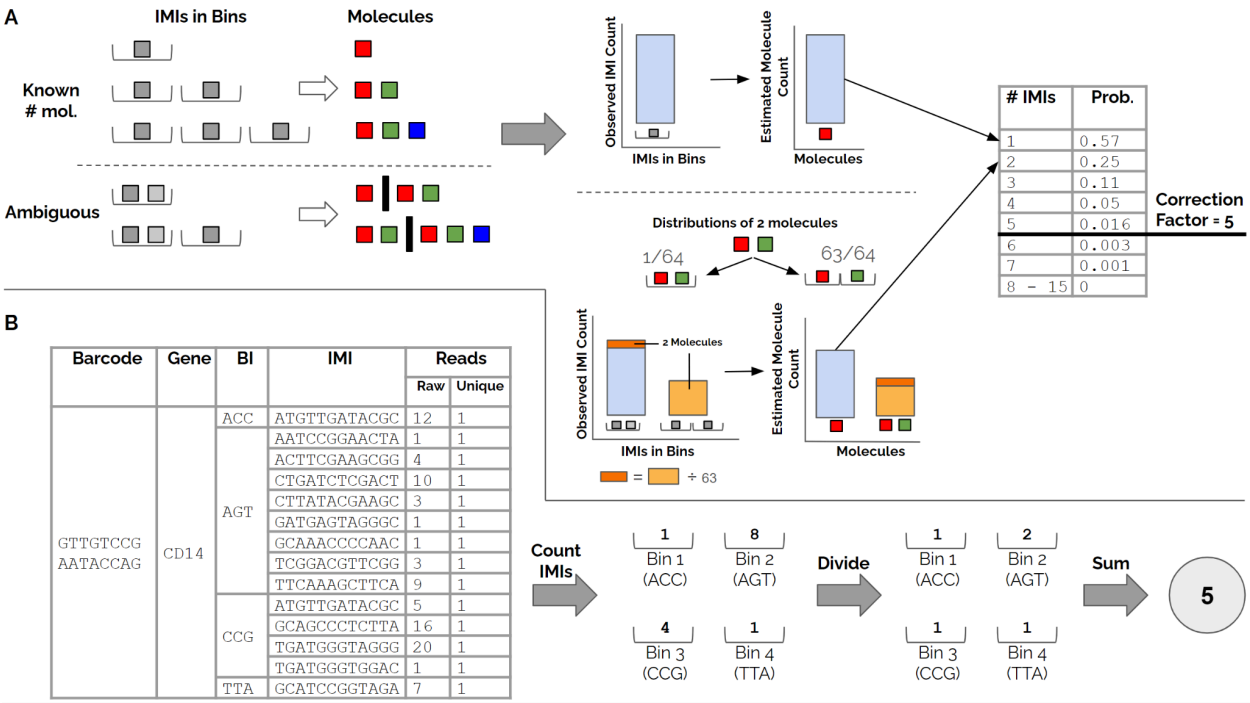

Figure 3: Molecular counting for PIPseq V chemistry. A: Estimating inflation and setting a correction factor. Left: examples of known and ambiguous molecular counts for different distributions of IMIs in bins. Center: Using known molecular counts and combinatorial possibilities of the 3-base binning index to estimate the counts of a single molecule inflated to different numbers of IMIs. This example illustrates the process for 1 IMI (no inflation) and 2 IMIs. The count of 1 molecule inflated to 2 IMIs is

corrected by subtracting 1/63 of the count of 2 IMIs in 2 bins. Right: The estimated counts lead to a set of inflation probabilities. We select a correction factor such that the cumulative inflation probability exceeds 99%. This means that at most 1% of counts will be inflated. B: Molecular counting. Within each barcode and gene, IMIs are sorted into bins and collapsed into unique counts. The unique counts are summed and divided by the correction factor. The estimated count for the barcode and gene is the sum of counts from all the bins.

Multi-mapped Reads

PIPseeker does not allow fractional read counts, and every molecule contributes a full count to a single gene-barcode combination. However, PIPseeker uses a pseudo-multi-mapping approach: If a molecule is mapped to multiple genes, one of the genes will be chosen randomly, with weighted probabilities based on the frequency of mapping to each gene. Consider the following examples:

- A molecule includes a single read, mapped to both gene 1 and gene 2. The count matrix will include this molecule with one of those genes, randomly selected with equal probability.
- Multi-mapping with deduplication: a molecule for which 5 copies (reads) mapped to both gene 1 and gene 2, and 5 other reads mapped to both gene 1 and gene 3. The total read count will be 10 for gene 1, 5 for gene 2, and 5 for gene 3. Consequently, the molecule will be assigned (and contribute a single count) to gene 1 with a 50% probability, gene 2 with a 25% probability, or gene 3 with a 25% probability.

Outputs

The raw matrix is contained in the files: `matrix.mtx.gz`, `barcodes.tsv.gz` and `features.tsv.gz`, all inside `<output path>/raw_matrix`. The format of the matrix is compatible with standard downstream analysis tools, e.g., *Seurat*.

Along with the raw matrix files, PIPseeker outputs a file, `<output path>/metrics/matrix_stats.csv`, which includes information from the molecular counting process:

- `mapped_transcriptome`: Number of reads mapped to the transcriptome (including or excluding introns, depending on whether `--exons-only` was used).
- `mapped_genome`: Number of reads mapped to genome locations not annotated as transcripts.
- `duplication_rate`: The ratio of mapped reads to total molecular counts.
- `sequencing_saturation`: A measure derived from duplication rate, which reflects the fraction of library complexity that was sequenced.
- `Invalid_index`: The number of molecular indexes that contain "N" bases.
- `mapped_intronic`: The number of reads mapped to introns if (`--exons-only` was used).

Cell Calling

After obtaining the count matrix, barcodes are separated into putative cell-containing PIPs and background PIPs, i.e., PIPs that did not capture a cell. This separation is based on the barcode rank plot (often referred to as the "knee plot"), which orders barcodes based on the number of transcripts associated with them (Fig. 4).

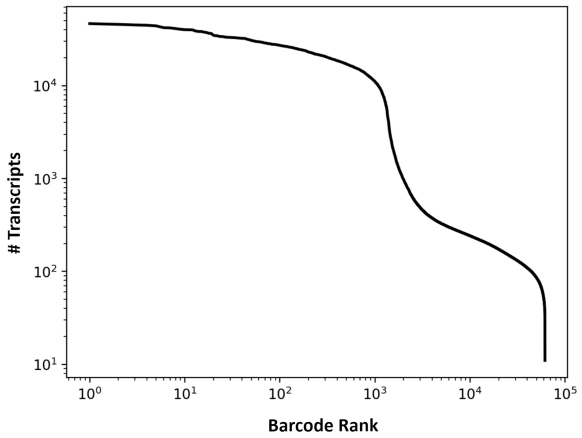

Figure 4: Barcode rank plot.

Typically, the cell barcodes are concentrated at the top mode of the rank plot, whereas the background barcodes are concentrated in the low transcript count mode. Therefore, the purpose of cell calling is to find a point in the first "knee" area that separates the two modes. This is commonly done by dividing the number of transcripts at the top of the rank plot by a constant denominator (e.g., 10). The problem with this approach is that it is highly sensitive to the shape of the rank plot, which varies strongly between different sample types and can be impacted by multiple experimental factors (e.g., the viability of cell input). Fig. 5 shows the result of calling cell barcodes up to 1/10 of the top barcode's count for HEK/3T3, PBMC and breast tissue samples. The cell portion is highlighted in green.

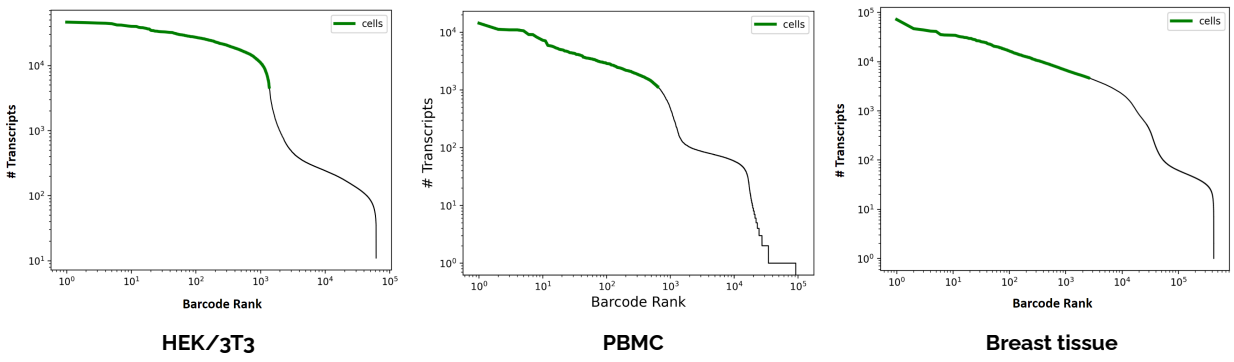

Figure 5: Cell calling up to a constant depth (1/10 of top barcode) for three different sample types of varying quality.

Cell calling for HEK/3T3 seems consistent with a visual inspection of the barcode rank plot: the entire high-transcript mode is captured. In the cases of lower quality PBMCs and breast tissue, however, the number of cell barcodes is clearly underestimated. This is because the top mode has a more negative slope and is less clearly defined than in the HEK/3T3 case.

To accommodate for differences between samples and facilitate accurate cell calling, PIPseeker performs cell calling at different sensitivity levels using the following steps:

1) Select a starting barcode, S, close to the top of the rank plot. To avoid selecting an outlier, PIPseeker detects the first barcode where the transcript count does not drop by more than 10% from the previous neighboring barcode on the plot.

2) Let M be the number of transcripts for barcode S. For each cell calling sensitivity level L_i, find a number of transcripts, T_i, such that:

$$T_L = M * 10^{-\left(0.5+0.25L_i\right)},\ L = \{min.\ sensitivity,\ \dots\ ,\ max.\ sensitivity\}$$

3) For each cell calling sensitivity level L_i, select all barcodes with a transcript count higher than or equal to T_L as cell-containing barcodes.

PIPseeker will default to five sensitivity levels (1 → 5), but the user can define the range of sensitivity levels using the `--min-sensitivity` and `--max-sensitivity` arguments. Fig. 6 shows cell calling at sensitivity levels 1 through 5 for a PBMC sample.

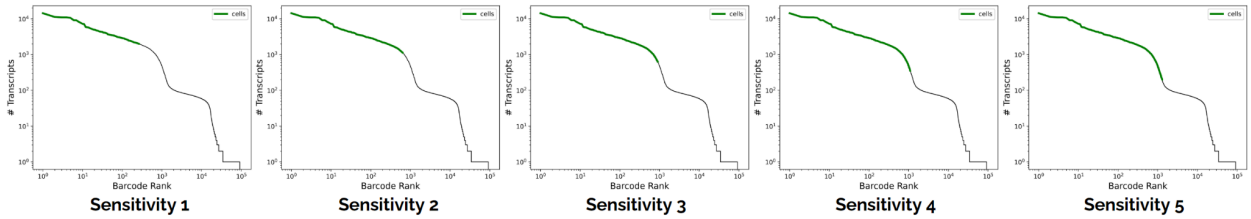

Figure 6: Cell calling with different sensitivity levels. The higher the sensitivity, the deeper down the rank plot barcodes are labeled as cells.

Cell calling inherently involves a tradeoff between the number of cells and their quality. One can arbitrarily increase the number of cells by calling deeper into the rank plot, but this increases the likelihood of selecting low-RNA content or otherwise compromised cells. The optimal number of cells recovered depends on the sample type and purpose of each experiment. For example, if some cell types in a sample (such as neutrophils or red blood cells) have low RNA expression, they may be present in high numbers in the lower transcript region of the plot. If those cells are of high interest., high-sensitivity cell calling may be warranted. On the other hand, if gene expression accuracy is a higher priority, it would be preferable to use low-sensitivity cell calling. Cell calling at multiple, different sensitivity levels allows users to observe a wide range of possible outcomes and determine the appropriate cell calling sensitivity for their data.

Outputs

After cells are called, a directory is created for each sensitivity level inside `<output path>/cell_calling`, containing a text file with the indices of the barcodes selected as cells and an image with a barcode rank plot. The indices start at 1 and correspond to the position of the barcodes in the file `barcodes.tsv.gz` of the raw matrix. Additionally, a filtered count matrix is generated for each sensitivity level and can be used for downstream analysis. Filtered matrices are stored in

`<output path>/filtered_matrix` and contain only the cell-associated barcodes for each sensitivity.

Manual Selection of Number of Cells

In some cases users may want more precise control of the number of called cells, beyond what is afforded with the five sensitivity levels. The `--force-cells` argument allows users to specify a desired number of cells, in which case this exact number of barcodes is selected from the top of the barcode rank plot. The outputs described above will be generated for the selected set of barcodes using the prefix force_*N*, with *N* being the number of cells input by the user.

Clustering

After selecting cell barcodes, PIPseeker performs a clustering analysis in order to quantify and visualize heterogeneity within the cell population and identify different cell types. For every set of called cells (different sensitivities or a number forced by the user), PIPseeker creates clustering maps using Leiden (graph-based) clustering, which automatically determines the number of clusters based on nearest neighbors. The user can also add K-means clustering with a specified range of values for K. Clustering maps are saved as UMAP (Uniform Manifold Approximation and Projection) plots, a popular method for projecting multi-dimensional data onto a 2-D space

Preprocessing Steps

The starting point for clustering is the filtered count matrix, in which every column represents a cell and every row represents a gene. A number of processing steps are performed before the actual clustering algorithm is run:

1) Cell normalization: Normalize each cell's data to unit norm, so that cells with similar expression profiles but different relative RNA abundance can still be clustered together.
2) Log transform: Transform the matrix using the $ln(x + 1)$ function in order to bring it closer to a normal distribution.
3) Highly-variable gene selection: Select a subset of genes with high variance to maximize the efficiency of clustering. For every gene, the variance in expression across cells is calculated. The genes are then ranked by variance, and the top genes at the N^{th} percentile are selected, with N being an optional user input specified using the `--clustering-percent-genes` argument.
4) Scaling: Scale feature columns (genes), so that each column has a mean of 0 and a standard deviation of 1.
5) PCA: Use Principal Component Analysis to reduce the dimensionality of the data from the scaled set of highly variable genes to a smaller number of components that best account for variance in the expression data. The number of components can be set automatically, as detailed below.

Following the preprocessing steps, clustering is performed in the new PCA space.

Automatic Determination of Clustering Parameters

PIPseeker determines optimal values for the number of neighbors used in k-nearest-neighbors (KNN) graph building, and for the Leiden algorithm resolution. Those parameters are derived from the number of cells, C:

$$neighbors = 2.7 * log(C) - 10.21$$

$$resolution = \frac{neighbors}{4*log(C) - 19.25}$$

If there are not enough cells available to calculate resolution (fewer than 200), a default value of 2.5 is used.

PIPseeker also determines the number of principal components automatically using an "elbow" method, choosing the minimum value of principal components n such that

$$\frac{ev_{n+1} - ev_n}{ev_1 - ev_{200}} < ev_{200}$$

where ev_n is the explained variance of the nth principal component.

If less than 200 components are available, the number of components used is the minimum of the number of cells – 1, number of genes – 1, and 15 components.

To override automatic selection of clustering parameters, PIPseeker requires the user to specify `--nearest-neighbors` for the KNN graph and `--resolution` for the Leiden algorithm. Users may also set the number of components used in PCA with `--principal-components`.

Clustering Sensitivity

PIPseeker supports clustering with sensitivities 'low', 'medium', and 'high', which can be set using `--clustering-sensitivity` (note that clustering sensitivity cannot be set when overriding automatic selection of clustering parameters). The number of clusters detected by Leiden will increase with clustering sensitivity. By default, PIPseeker will cluster using 'medium' sensitivity, where clustering parameter values are calculated using the previously described optimized equations. In 'low' sensitivity mode, the value of the resolution parameter in Leiden is decreased by 40% and the value of the nearest neighbors parameter in k-nearest-neighbors graph building is increased by 40% compared to the default 'medium' sensitivity values. 'high' clustering sensitivity increases the value of the resolution parameter by 40% and decreases the value of nearest neighbors by 40% compared to the 'medium' sensitivity values.

In 'low' sensitivity mode, fewer communities are detected, resulting in fewer overall clusters which contain more cells. Conversely, 'high' sensitivity results in more communities overall and allows for smaller possible community size.

Graph-Based Clustering

PIPseeker performs clustering using the Leiden algorithm, the current standard for graph-based clustering in scRNA-seq datasets. A k-nearest-neighbor (KNN) graph is constructed from the PCA

matrix, which is converted to a directed node-edge graph and passed into Leiden. The Leiden algorithm uses a modularity optimization of graph communities to determine ideal cluster membership. Leiden has the added advantage of minimal user inputs (e.g., number of clusters, cutoffs, etc.) compared to other methods. As noted above, the user can control some of the clustering parameters (`--nearest-neighbors`, `--resolution`) or have them be determined automatically.

K-Means Clustering

PIPseeker also uses the popular K-means algorithm to divide cells into distributed clusters. It runs the algorithm for different values of K, representing a predetermined number of clusters. The minimum and maximum of the range of K values can be controlled by the user (`--min-clusters-kmeans` and `--max-clusters-kmeans`) and should be tailored to the number of cell types expected to be present in the sample.

Differential Expression

Once clustering is completed, PIPseeker finds the differentially expressed genes associated with each cluster. For this step, PIPseeker uses the Wilcoxon rank-sum test for independent variables (Mann-Whitney U). This nonparametric test is used to determine whether two samples come from the same distribution by ranking the values associated with two groups together, and calculating the sum of those ranks within each group. Those ranks are used to calculate a test statistic (U), which PIPseeker uses to create a rank order of differentially expressed genes for each cluster.

For each cluster, PIPseeker computes differential expression for genes associated with cells across that cluster vs. all other cells. The top genes, sorted by descending Z-score, are exported to a CSV file. The number of reported genes is controlled by the `--diff-exp-genes` argument. This process is performed for different values of K in K-means clustering, as well as for graph-based Leiden clustering.

Cell Type Annotation

For certain sample types, PIPseeker can assign cell types to clusters based on the similarity in gene expression to a known reference dataset. The user input `--annotation` points to a reference file, which can be downloaded from the Fluent BioSciences website or generated using PIPseeker buildannotref.

Cell types are assigned based on the correlation between differential expression scores for each cluster and the scores for the annotated types in the reference dataset If the correlation with a certain cell type is high (r >= 0.5), the cluster is assigned that type as “Primary”. If there is high correlation with another type (r >= 0.3), the second best-matching cell type gets assigned as a “Secondary”. The primary cell types are then used to generate a UMAP plot colored by cell types. Clusters with low similarity to all reference cell types are not assigned and are labeled “Unknown”.

Clustering sensitivity levels can be used in combination with cell type annotation to optimize clustering outputs. In the example in Fig. 7, 23,779 PBMC cells were clustered using ‘low’, ‘medium’, and ‘high’ clustering sensitivity modes. The number of clusters and detected cell types increase with clustering sensitivity. In ‘low’ sensitivity mode, 10 cell types are identified from 13 detected clusters. This mode best maintains the Naïve CD4 T cell cluster, spreading this cell type across three clusters,

compared to four clusters in 'medium' sensitivity mode and five clusters in 'high' sensitivity. In 'medium' sensitivity mode, an additional Central Memory CD4 T cell cluster is detected, Finally, 'high' clustering sensitivity reveals a cluster of rare stem cells (HSPC).

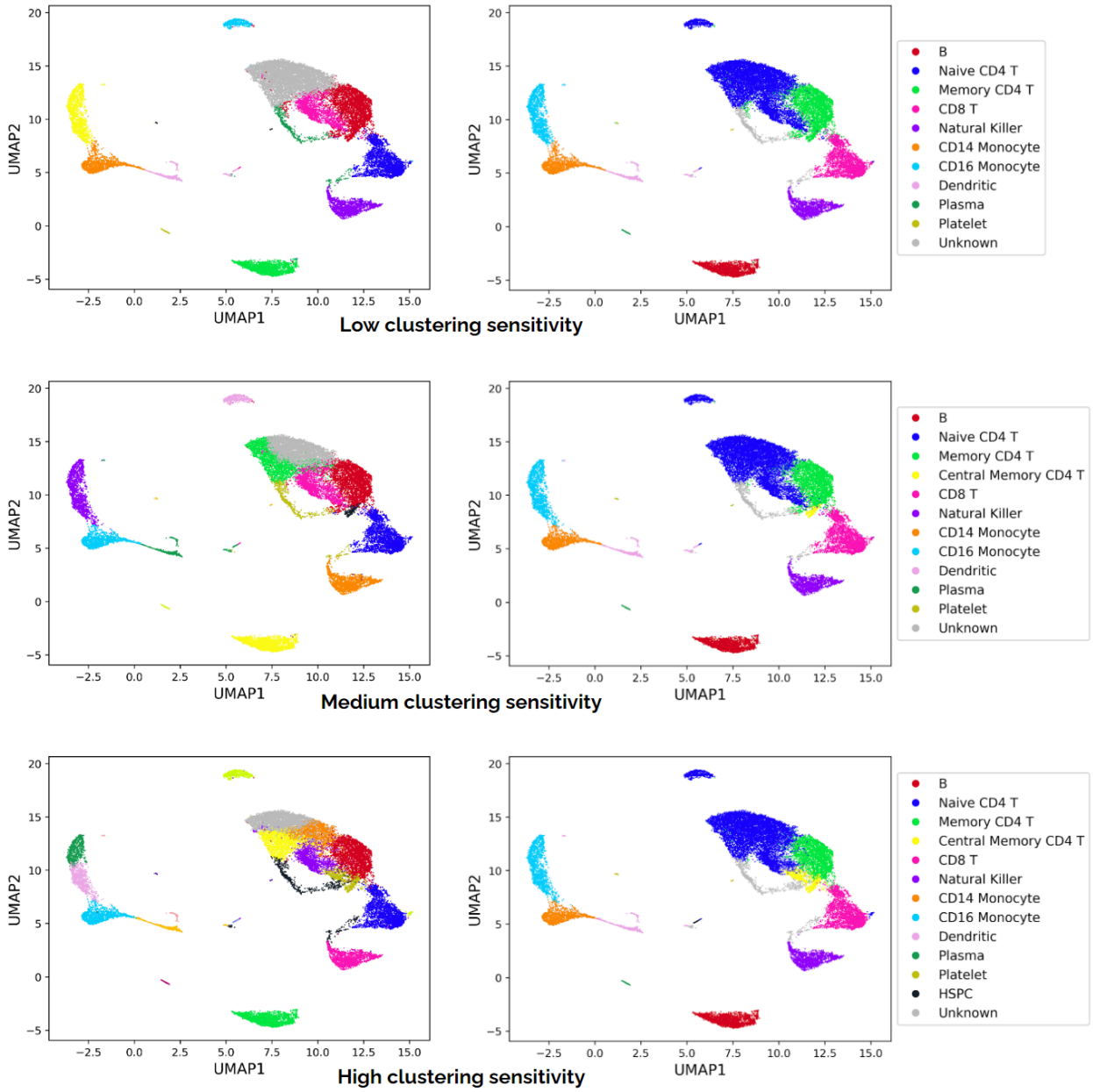

Figure 7: Graph-based clustering and cell type annotation for a PBMC sample at three different clustering sensitivities. The left panels show graph-based clustering results, and the right panels show the corresponding cell type annotations, as indicated in the legends. Note the increase in the number of clusters and in the detection of additional cell types as sensitivity increases. Also note that multiple clusters can get assigned the same cell type, and some clusters do not get assigned a cell type at all ("Unknown").

Increasing clustering sensitivity can help isolate clusters of rare cell types. On the other hand, high sensitivity leads to a higher number of clusters, which in turn can result in a less reliable gene expression profile for each cell type. How to balance this tradeoff depends on the specific purpose of each experiment.

Outputs

A directory is created for each cell calling mode inside `<output path>/clustering`, containing the following:

- `clusters.csv`: cluster assignment of each cell from graph-based clustering and from any K-means runs.
- `umap/*.png`: UMAP plots of clusters.
- `top_genes/*.csv`: top-ranked differentially-expressed genes for each cluster.
- `differential_expression/*.csv`: top-ranked differentially expressed genes for each cluster, with additional statistics for each gene: Z-score, p-value, adjusted p-value and log2 fold change.
- When cell type annotation is used:
    - `cell_type_annotation/<sample type>/cell_types/*.csv`: cell type assignments for each cell (barcode ID) and for each cluster.
    - `cell_type_annotation/<sample type>/umap/*.png`: cell type UMAP plots for each clustering type.

Metrics

After completing the analysis, various metrics are calculated and reported for each cell calling mode (sensitivity level or forced number of cells).

Basic Metrics

- Total number of reads in the input FASTQ files.
- Total number of reads above minimum length after R2 trimming.
- Total number of reads for which valid barcodes were detected (reads sent to *STAR* for mapping).
- Total number of reads mapped to the transcriptome.
- Percentage of transcriptome-mapped reads out of the total number reads provided to STAR after FASTQ processing (reads with valid barcodes).
- Percentage of reads in cells
- Percentage of reads associated with cell-containing barcodes out of the total number of input reads.
- Number of called cells
- Mean reads per cell: the total number of input reads divided by the number of cells.

- Duplication rate: The number of total mapped reads in cells divided by the number of unique (deduplicated) reads in cells. For example: If, on average, every molecular index is associated with three reads (two deduplicated reads per molecule), the duplication rate would be 3.
- % sequencing saturation: The percentage of the total cDNA library that has been observed at a given sequencing depth. Also indicates the probability that when sequencing an additional read, the library molecule would have already been detected. Defined as:

$$100 \star (1 - \frac{1}{dup.rate})$$

- Total number of transcripts in cells
- Median transcripts per cell, based on the distribution of counts of unique deduplicated indexes (transcripts) associated with each cell barcode.
- Number of genes expressed in cells: This number includes all genes with a nonzero count in the filtered matrix, and gives equal weight to every gene regardless of expression level.
- Median genes per cell
- Percentage of mitochondrial transcripts in cells: This is based on gene names that include prefixes like "MT-" and may not work with every mapping reference.

Barnyard Metrics and Estimation of Multiplet Rate

One common way to assess the quality of scRNA-seq data is a combined species ("barnyard") experiment, typically involving a mixture of human and mouse cells. Such an experiment can provide information about the prevalence of multiplets, i.e., PIPs that capture more than a single cell. It can also assess the degree of interference from ambient RNA by measuring cross-species contamination. The more human genes found in mouse cells and vice versa, the more ambient RNA contamination is likely present in the system.

When the reference transcriptome is a combined human and mouse transcriptome, and the `--run-barnyard` flag is used, PIPseeker can provide an additional set of metrics specific to a barnyard experiment:

- Number of human cells: the number of cell barcodes with at least 85% of the transcripts mapping to the human reference.
- Number of mouse cells: the number of cell barcodes with at least 85% of the transcripts mapping to the mouse reference.
- Number of multiplets: the number of cell barcodes with less than 85% of the transcripts mapping to the human or the mouse reference.
  - Note that "hidden" multiplets can also occur when two or more cells of the same species are captured in a single droplet. Currently, PIPseeker only reports the number of observed multiplets and does not include a prediction for hidden multiplets that may be present in the sample.
- Total number of transcripts in human cells
- Median human transcripts per human cell
- Number of human genes expressed in human cells

- Median human genes per human cell
- Percentage of mouse transcripts in human cells
- Total number of transcripts in mouse cells
- Median mouse transcripts per mouse cell
- Number of mouse genes expressed in mouse cells
- Median mouse genes per mouse cell
- Percentage of human transcripts in mouse cells

An important output of a barnyard analysis is called a barnyard plot (Fig. 8). It shows the number of mouse transcripts vs. the number of human transcripts, with different colors indicating human cells, mouse cells and multiplets:

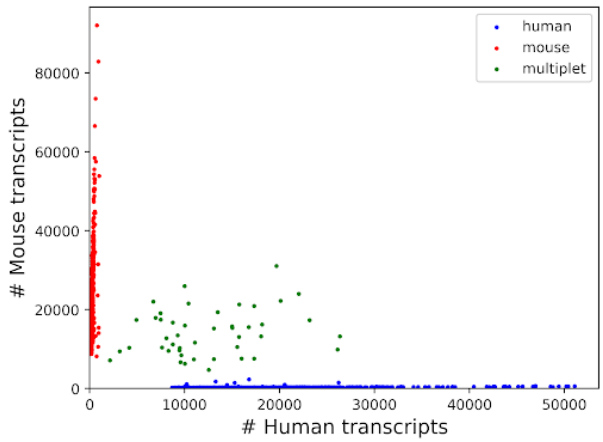

Figure 8: "Barnyard" plot.

Outputs

After all the metrics are calculated, results for each cell calling mode are reported in the following files:

- Basic metrics are stored in csv and json formats in `<output path>/metrics`.
- If applicable, barnyard metrics are stored in csv and json formats in `<output path>/barnyard`.

Synthetic Nucleotide Tags

In addition to gene expression based on sequences derived from captured mRNA, PIPseeker can also process sequencing data from synthetic nucleotide tags (SNTs) that have been introduced to label or modify selected targets in a sample. The SNTs used and methods for modifying unique samples with such tags is highly dependent on user needs. PIPseeker provides a generic solution to enable analysis of gene expression in conjunction with SNT counting. This technique may be used for a variety of applications including, but not limited to, applications that measure cell-surface protein

expression or guide-RNA perturbations, and cell hashing for sample multiplexing and multiplet detection.

SNTs are sequenced separately from the cDNA sample. While SNT targets are identified using tiered cell barcodes in exactly the same way as target gene expression, the FASTQ data is not mapped to a reference genome. Instead, reads are matched against a file containing the SNT sequences. Reads are then deduplicated and counted to produce count matrices similar to those derived from gene expression data.

Tags used for sample multiplexing are named Hashtag Oligonucleotides (HTOs) and can be processed in PIPseeker independently or at the same time as SNTs. The initial processing is similar, but some outputs are specific to each technique.

FASTQ Processing

After completing the gene expression analysis, PIPseeker can process additional FASTQ files provided using `--snt-fastq` and/or `--hto-fastq`. It matches reads in R2 to sequences in files specified by `--snt-tags` and/or `--hto-tags`. Tag files are CSV format files (no header), with columns for sequence, tag ID and (optionally) a tag description, as in the example below:

```
ACTGTA,Tag_1,Tag 1 description
CCTGTA,Tag_2,Tag 2 description
CACCGT,Tag_3,Tag 3 description
ACGCTA,Tag_4,Tag 4 description
TCAACT,Tag_5,Tag 5 description
```

Cell barcodes are matched in a tiered fashion just like the RNA data. With PIPseq v4, the molecular indexes are then matched with an error tolerance of 1. Each index is collapsed into the most commonly occurring index from which it is one base apart, if such indexes exist. Otherwise, the index is retained as a distinct molecular identifier. Finally, the R2 reads are matched against the sequences in the tags file, again with an allowed Hamming distance of 1.

With PIPseq V, molecular counts are read-based and there is no deduplication process. The SNT/HTO count matrix reflects the count of raw reads for each barcode+gene combination.

Raw Count Matrices

The FASTQ processing step results in a raw count matrix for the tags. It is similar to a gene expression count matrix, only here the features are SNTs or HTOs. This matrix is then merged with the RNA raw matrix to produce a single matrix containing both genes and tags as features. The barcodes in the tags matrix are renumbered to match those of the RNA matrix, and those that are unique to the tags matrix are appended after the RNA matrix's barcode list. The tags in the feature file are appended after the genes to produce a merged features file.

The following raw matrix outputs are generated in the output directory:

- SNT: `SNT/snt_raw_matrix`, `SNT/raw_matrix` (merged)
- HTO: `HTO/hto_raw_matrix`, `HTO/raw_matrix` (merged)

- If both SNTs and HTOs are used, the merged HTO matrix will include RNA features (genes) and SNT features.

Filtered Count Matrices

Like in gene expression analysis, the merged raw matrices are subset to produce filtered matrices that only contain cell barcodes. The cell barcodes are the same ones selected in the cell calling step based on RNA data. The filtered matrices are stored in `SNT/filtered_matrix` and/or `HTO/filtered_matrix`.

Output Metrics

- Total number of reads in the input FASTQ files.
- Total number of reads for which valid barcodes were detected.
- The number of reads with valid barcodes that were successfully matched to one of the tags.
- Duplication rate (v4 only, see description for RNA-based analysis).
- % sequencing saturation (v4 only, see description for RNA-based analysis).
- Total deduplicated (v4) or raw (V) read count in cells
- Median count of reads in cells
- Median number of unique tags in cells
- % Tagged cells (cells with at least one detected tag
- Median count of reads in tagged cells
- Median number of unique tags in tagged cells

Metrics are saved for each cell calling mode in `SNT/metrics` or `HTO/metrics`. In addition, a text file with the total count for each tag is saved in a directory called `counts_per_tag`.

Tag Detection Rank Plots

In addition to standard RNA barcode rank plots, PIPseeker produces rank plots that show the effectiveness of tag detection for each modality, SNT and/or HTO. The rank plot lines are segmented in regular intervals on a log scale. The color of each segment is determined by the percentage of barcodes with at least one tag. If fewer than 20% of barcodes have any detected tags, the segment will be white. Otherwise, the segment will be colored based on the scale shown in the plot's colorbar. The cell fraction for each selected cell calling sensitivity is located to the left of the vertical dashed blue line. Ideally, barcodes in the cell fraction should be colored by the maximal value on the color scale and non-cell barcodes should be white, with no detected tags. The plots are saved in `SNT/plots/barcode_rank` or `HTO/plots/barcode_rank`.

Feature Plots

The tag counts are normalized using center-log ratio (CLR) transformation. A color gradient representing tag expression is overlaid in UMAP space with locations based on the results of cell clustering. The plots are saved in `SNT/plots/feature` or `HTO/plots/feature`.

Samples without Gene Expression Data

PIPseeker can process SNT data without an accompanying RNA count matrix. If SNT data is input alone, samples will be directed through a separate pipeline, which is similar to the RNA workflow but has some key differences.

Cell calling is performed on the SNT raw matrix instead of the RNA raw matrix. Sensitivity levels are still utilized, resulting in called cells based solely on the shape of the SNT rank plot. The nature of SNT libraries lends itself to less separation between signal and background, which may require a higher sensitivity level (adjusted using `--max-sensitivity`) to capture the appropriate number of cells.

Unlike the RNA-based workflow, no feature selection is performed before clustering, so all SNTs are retained. The other preprocessing steps prior to clustering are similar. The clustering step is identical to the RNA processing, performing both graph-based and k-means clustering. In addition, for each SNT, a threshold is identified to establish whether a cluster is positive or negative for the SNT. This is based on identifying, for each SNT, two populations across all cells using 1-dimensional Kmeans clustering, then using the mean of the cluster centers as the positivity threshold. If the mean expression in a cluster is above the detected threshold, the cluster is considered positive for that SNT. This information is stored in `SNT/clustering` for each cell calling mode and clustering mode in directories named `top_features`.

The cell type annotation strategy is based on the marker positivity metric described above. This process is performed for all available SNTs to generate a matrix of clusters and SNTs. The matrix of positive and negative values is then compared to an SNT annotation reference. The SNT annotation is a CSV file in the following format:

```
cell_type,<marker 1>,<marker 2>,<marker 3>,<marker 4>,...
<type 1>,1,0,0,1,...
<type 2>,0,1,1,0,...
...
```

The SNT annotation describes which markers are expected to be positive for each cell type. Cell types are assigned in the same way as the RNA annotation, except using the Jaccard index instead of a Pearson correlation, with Primary and Secondary cell types being assigned based on a minimum similarity and differential.

SNT feature plots are generated for each SNT. The SNT expression is overlaid on the SNT UMAP generated during clustering.

If cell hashing is included in an analysis without RNA data, a merged SNT + HTO raw matrix will be created, but none of the subsequent steps (e.g., clustering) will be performed.

Cell Demultiplexing with Hashing

Cell hashing is typically used to combine distinct samples into a single run and later separate them based on different barcoded sequences attached to each sample. Separating, or demultiplexing, the samples involves building a negative distribution for each HTO in order to define a threshold of

expression beyond which a cell can be confidently assigned to a sample. Each HTO is assigned in this manner to a single sample or determined to match more than one sample (multiplet) or none at all.

Building the negative distribution

To define a negative distribution for each hash, K-means clustering is performed on the CLR-normalized HTO matrix. The number of clusters is set as one more than the number of hashes in order to produce an "unassigned" cluster in addition to a cluster for each hash. For each hash, the array of normalized expression values is separated into distributions associated with each cluster. This includes the unassigned cluster, which presumably contains all cells that do not have a putative label. The cluster with the lowest average expression for that hash is termed the *negative* distribution. The 99th percentile of that negative distribution becomes the threshold for determining whether a cell is positive or negative for that hash. Normally, the negative distribution will be the unassigned cluster, but sometimes it can be another hash distribution if the unassigned cluster has elevated expression.

PIPseeker outputs "ridge" plots (Fig. 9A) for each hash with the normalized expression value distributions of each cluster (row in the plot) and the threshold defined from the negative distribution. Plots for each hash are saved as in `HTO/plots/ridge`.

After performing the same analysis for each hash, the total number of *positives* are calculated for each cell. If a cell is positive for no hashes, it is determined to be in a global negative category. If a cell is positive for more than one hash, it is determined to be in a global multiplet category. Most cells should be positive for only a single hash and placed in a global singlet category.

Detecting multiplets with hashing

As noted above, some cells receive a "multiplet" designation. It is important to note, however, that hashing cannot determine whether these cells have more than one cell's worth of material. It simply means that a cell has higher expression than the negative distribution threshold in more than one hash, which is also consistent with a global *ambiguous* category. When cells are not labeled well with hashes, the "multiplet" category is often highly represented by ambiguous calls because of the low signal-to-noise ratio. In a successful hashing experiment, most of the cells should be positive for only one hash, putting them in a global singlet category. In this case, the singlet global category can be further separated into each hash.

Diagnostic Outputs

To visualize the cell distribution, PIPseeker generates a heatmap (Fig. 9B) of CLR-normalized expression with column values sorted into singlet, multiplet, and negative global categories (left to right), and separated by black lines. The singlet order is further separated into the order of the assigned hashes (top to bottom), and sorted by expression of the assigned hash. Multiplets and negatives are sorted by total expression across all hashes. A color legend is appended to the top of the heatmap, with colors corresponding to the hash names on the y-axis. The white bar represents multiplets, while the gray represents negatives. The heatmap for each cell calling mode is saved in `HTO/plots/heatmap`.

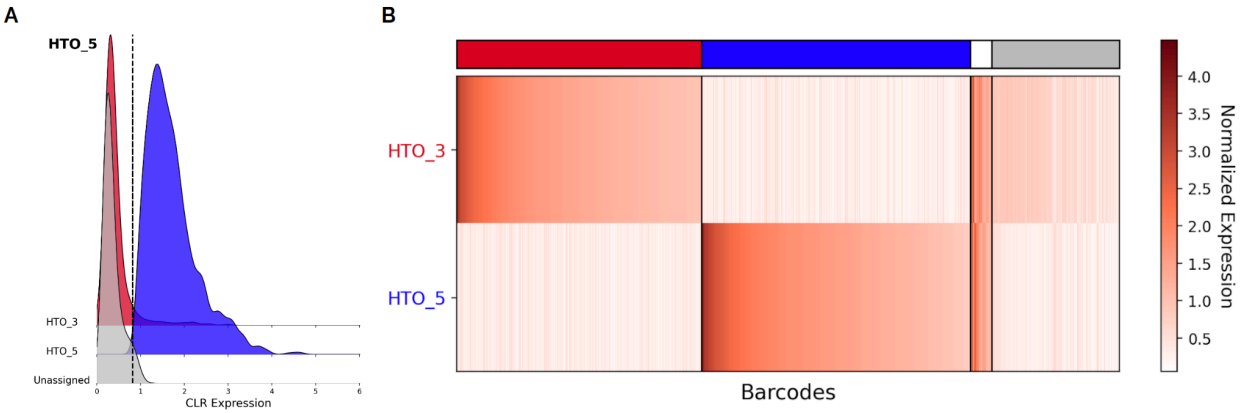

Figure 9: HTO output examples. A: Ridge plot for a single hash. B: Hashing heatmap. Multiplets are denoted by a white rectangle and the negative distribution is denoted by a gray rectangle.

PIPseeker outputs a CSV file for each cell calling mode in `HTO/demux_data` with the following information associated with each cell:

- Hash global classification: singlet, multiplet, or negative
- Hash ID: hash ID, multiplet, or negative
- Max ID: hash associated with maximum CLR-normalized expression, regardless of PIPseeker hash global classification
- Hash max: the maximum CLR-normalized expression value
- Hash second max: the second highest CLR-normalized expression value
- Margin: the difference between the max and second max
- Heatmap order: position (from left) in the heatmap

Analysis of demultiplexed Matrices

After demultiplexing, PIPseeker divides the RNA matrix into separate matrices for each singlet hash identity in `HTO/demux/<HTO name>`. Those matrices are then used for further analysis of each demultiplexed sample. PIPseeker creates clustering outputs for each demultiplexed sample, using the same clustering parameters that were used for the combined sample. If SNT data is included, SNT feature plots will be generated for each demultiplexed sample. The outputs for each demultiplexed sample are included in a separate section of the analysis report.

The user may want to reprocess a demultiplexed sample with different parameters than were used for the combined sample. Some examples are increasing the clustering sensitivity, adding K-means clustering, or changing the SNT colormap in order to have a specific color for each sample. Use the `--hash` input with PIPseeker cells to reanalyze a single demultiplexed sample.

Analysis Reports

Every run of PIPseeker produces a report in HTML format, `<output path>/report.html`, which include basic metrics, a barcode rank plot, a clustering map, a differential expression table, and, if

applicable, a cell type annotation map. If "barnyard" mode is used, those statistics and a barnyard plot will be included. If SNT or HTO data were included in the analysis, those results will also be incorporated into the report.

The report contains a tab for each cell calling sensitivity level, as well as any analyses performed with a forced number of cells. The user can browse through the different sensitivities and select the optimal sensitivity level or decide on a number of cells to force (`--force-cells`) in a subsequent reanalysis (PIPseeker cells).

In the case of an analysis with a manually selected (forced) number of cells, a report will be generated for this specific number, and a new tab will be added to the combined report.

Running PIPseeker

PIPseeker can run on Linux, Mac and Windows. A Quick Start tutorial is available here: https://www.fluentbio.com/resources/pipseeker-tutorial

System Requirements

- The required hardware settings for running PIPseeker depend on the size and characteristics of the input data and cannot be predicted precisely ahead of time. A good rule of thumb, however, is to allocate the following for every 1 GB of input FASTQ (in .gz format):
    - 1GB of RAM
    - 2.5GB of free disk space.

  In addition, PIPseeker requires 3GB of RAM for every additional processing thread (specified using `--threads`), and at least as much free memory as the size of the genome reference.
- PIPseeker runs a resource check before starting the analysis (It can be turned off using `--skip-preflight-check`).
- PIPseeker has been tested on Ubuntu 20.04 & 18.04, CentOS 7.9-2009, Windows 10 and macOS 12.3.1.
- The Windows version of PIPseeker runs from a Docker container and requires a local installation of Docker (https://docs.docker.com/get-docker). The PIPseeker executable automatically pulls the image from the Docker repository and runs the analysis from a container.
    - Note that Docker requires a Linux backend installed on your machine. Please see this page for more information: https://docs.docker.com/desktop/install/windows-install

Setting up and Running the PIPseeker Executable

- Download and unpack the PIPseeker package and move it to a permanent location.

- On Mac and Linux, make sure the PIPseeker file has executable permission. After uncompressing the downloaded archive and navigating to the uncompressed directory, you can change its permissions using: `chmod +x pipseeker`
    - Mac users may need to provide the system additional permissions before the executable can be run.
- If you downloaded a mapping reference from the Fluent BioSciences website, make sure it is uncompressed (both the .tar and the .gz components). The reference should be a directory.
- On Windows, run PIPseeker from a Command Prompt or from PowerShell. PIPseeker is not a clickable GUI application.

Running PIPseeker from a Docker Container

- Docker images are available on the downloads page and can be used to run PIPseeker on any OS.
- When running in a virtual environment using the Docker image, make sure the virtual machine's resources are not limited in the Docker client settings.
- Make sure to mount any directory used as an input argument, as well as any directory containing a file used as an input argument.
- If running on Windows and any of your input, index or output directories are on a network drive, you may need to explicitly allow that drive to be mounted using the Docker client settings.

Running the complete analysis workflow: PIPseeker full

PIPseeker full is the main mode of operation and will typically be the first analysis to be launched on a sample after sequencing.

Examples

Show the full list of arguments:

```
$ <path to PIPseeker>/pipseeker full --help.
```

Standard gene expression analysis without changing any of the default arguments:

```
$ <path to PIPseeker>/pipseeker full --fastq <FASTQ files prefix>
--star-index-path <path to mapping reference> --chemistry <PIPseq chemistry
version> --output-path <path to output directory>
```

Gene expression analysis with high-sensitivity graph-based clustering and K-means clustering with 4 or 5 clusters:

```
$ <path to PIPseeker>/pipseeker full --fastq <FASTQ files prefix>
--star-index-path <path to mapping reference> --chemistry <PIPseq chemistry
version> --output-path <path to output directory> --clustering-sensitivity high
--min-clusters-kmeans 4 --max-clusters-kmeans 5
```

Gene expression + SNT analysis, with SNTs starting at the 5th base in R2 :

```
$ <path to PIPseeker>/pipseeker full --fastq <FASTQ files prefix>
--star-index-path <path to mapping reference> --chemistry <PIPseq chemistry
version> --output-path <path to output directory> --snt-fastq <SNT FASTQ file
prefix> --snt-tags <path to SNT tags CSV file> --snt-position 4
```

Input Arguments

General

`--chemistry` (string)

Version of the PIPseq assay (v3, v4, or V).

`--output-path` (string)

The directory where all PIPseeker outputs will be stored. If not specified, will be set to the same directory as the directory containing the input FASTQ files.

`--verbosity` (integer, default: 1)

Verbosity level: 0 (silent), 1 (concise), or 2 (detailed).

`--skip-version-check`

Do not check for newer versions of PIPseeker on the Fluent BioSciences website.

`--skip-preflight-check`

Skip the system checks for sufficient resources prior to data processing.

`--resume-last-run`

Resume processing from the last completed step in the output directory, instead of restarting from the raw FASTQs. This is useful for recovering from a run that failed for any reason, such as running out of disk space or losing power.

`--threads` (integer, default: 0)

Number of threads to use for parallel processing. Allowed values:

- Exact number of threads

- 1 for no parallel processing
- 0 to auto-detect and utilize all available cores

Note: the number of threads is capped at 16 for the FASTQ processing step, and at 32 for all other steps .

`--random-seed` (non-negative integer, default: 0)

Seed for random number generation. This affects FASTQ downsampling, raw matrix generation (when resolving multi-mapping) and clustering.

`--dpi` (integer, default: 200)

DPI (dots per inch) resolution for all saved figures.

`--save-svg`

Save copies of all generated plots in Scaled Vector Graphics (SVG) format. By default, PIPseeker only saves plots in Portable Network Graphics (PNG) format.

FASTQ Processing

`--fastq` (string)

The path to the source directory for input FASTQ files and a prefix for the names of the files to be processed. Replace the prefix with a period (`.`) to indicate that all files in the directory should be processed.

For example, consider a situation where the input directory contains two-lane results from two different samples:

```
$ ls my_fastq_dir

sample_1_L001_R1.fastq.gz      sample_1_L001_R2.fastq.gz
sample_1_L002_R1.fastq.gz      sample_1_L002_R2.fastq.gz
sample_2_L001_R1.fastq.gz      sample_2_L001_R2.fastq.gz
sample_2_L002_R1.fastq.gz      sample_2_L002_R2.fastq.gz
```

Use `--fastq my_fastq_dir/sample_1` and `--fastq my_fastq_dir/sample_2` in two separate runs to include only the relevant files for each sample.

Note: R1 files must be at least 54 bases when using v4 chemistry and 45 bases when using V chemistry. Make sure to not trim any reads while demultiplexing data from the sequencer, e.g., when converting from BCL to FASTQ format.

`--downsample-to` (integer)

An integer representing the target number of reads to be included if downsampling is desired. Reads will be selected randomly and only the selected subset will be considered for barcode matching.

`--input-reads` (integer)

An integer representing the number of input reads if `--downsample-to` is specified. In order for PIPseeker to calculate the downsampling ratio, the total number of reads must be known in advance. `--input-reads` should include the total number of reads combined across all input files. If it is not specified, PIPseeker will count the lines in the input files, so using this argument can reduce running time.

`--retain-barcoded-fastqs`

A flag for retaining the intermediate, barcoded FASTQ files. If not specified, these files will be discarded after completion of the analysis. Note that the original input FASTQ files are always retained, regardless of this flag.

Mapping

`--star-index-path` (string)

The path to a directory containing a STAR reference to use for mapping. Some standard references are available to download from the Fluent BioSciences website. Custom references can be generated with PIPseeker buildmapref (see below).

`--remove-bam`

Remove the BAM file *STAR* output after run completion.

Molecular Counting

`--exons-only`

Restrict molecular counting to include only reads that map entirely to exonic regions.

Cell Calling

`--force-cells` (integer)

Force a specific number of barcodes to be considered as cells. If specified, this number of barcodes will be selected from the top of the barcode rank plot. This will override the default mode of cell calling using sensitivity levels.

`--min-sensitivity` (integer, default: 1)

The minimum sensitivity level for cell calling. This number sets the bottom end of the range of sensitivity levels.

`--max-sensitivity` (integer, default: 5)

The maximum sensitivity level for cell calling. This number sets the top end of the range of sensitivity levels. To call more cells, increase this number.

Barnyard Analysis

`--run-barnyard`

Enable “barnyard” metrics to be generated and included in the analysis report

Clustering

`--clustering-percent-genes` (float, default: 10)

Percentage of genes to use for clustering. Genes with the highest variability in expression up to this percentile rank will be included.

`--diff-exp-genes` (integer, default: 50)

Number of top differentially expressed genes to report for each cluster.

`--principal-components` (integer)

The number of components for PCA dimensionality reduction.

`--nearest-neighbors` (integer)

Number of nearest neighbors for graph-based clustering. This defines the minimum number of cells per cluster.

`--resolution` (float)

Resolution parameter for graph-based clustering.

Note: --principal-components, --nearest-neighbors and --resolution must all be used or omitted at the same time. You cannot specify one argument and leave the others unspecified. If none of the arguments are specified, PIPseeker will assign them automatically.

`--clustering-sensitivity` (low, medium or high, default: medium)

Sensitivity level for graph based clustering.

`--min-clusters-kmeans`, `--max-clusters-kmeans` (integer)

Minimum and maximum number of clusters for K-means clustering. A separate K-means analysis will be done for every number in the range. By default, no K-means clustering is performed.

`--umap-axes`

Add axis ticks and numeric labels to UMAP plots. If not specified, UMAP axes will remain blank..

Cell Type Annotation

`--annotation` (string)

File to use as reference for cell type annotation of the clustering results. Annotation reference files are available to download from the Fluent BioSciences website. Custom annotation references can be generated with PIPseeker buildannotref (see below).

SNT

`--snt-fastq` (string)

The path to the source directory for SNT FASTQ files and a prefix for the names of the files to be processed. Replace the prefix with a period (`.`) to indicate that all files in the directory should be processed.

`--snt-tags` (string)

A path to a CSV file containing the SNT sequences and labels.

`--snt-position` (integer or comma-separated list of integers, default: 0)

Starting position(s) of the SNT sequence in the R2 FASTQ file (0-based: the first base is at position 0).

`--snt-label` (string, default: SNT capture)

Label for SNT features in count matrices, e.g., "Antibody Capture", "CRISPR Capture"..

`--snt-no-plots`

Disable the generation of SNT feature plots. This may be desired when the number of tags is large.

`--snt-colormap` (string, default: gray-to-green)

Colormap for SNT feature plots. See appendix 1 for available colormaps.

`--snt-colorbar`

Include a colorbar in SNT feature plots.

`--snt-min-percent` (float, default: 1)

Bottom colormap limit for SNT feature plots, defined as a percentile rank.

`--snt-max-percent` (float, default: 99)

Top colormap limit for SNT feature plots, defined as a percentile rank.

`--snt-min-value`, `--SNT-max-value` (float)

Bottom and top colormap limits for SNT feature plots, defined as scalars. Note that scalars and percentile ranks cannot be used together in the same analysis.

`--snt-annotation` (string)

Path to a file to use as reference for cell type annotation of the SNT clustering results. The file describes which markers should be highly expressed (positive) for each cell type.

HTO

`--hto-fastq` (string)

The path to the source directory for HTO FASTQ files and a prefix for the names of the files to be processed. Replace the prefix with a period (`.`) to indicate that all files in the directory should be processed.

`--hto-tags` (string)

A path to a CSV file containing HTO sequences and labels.

`--hto-position` (integer, default: 0)

Starting position of the HTO sequence in the R2 FASTQ file (the first base is at position 0).

`--hto-colormap` (string, default: gray-to-red)

Colormap for HTO plots. See appendix 1 for available colormaps.

`--hto-colorbar` (string, default: gray-to-red)

Include a colorbar in HTO feature plots.

`--hto-min-percent` (float, default: 1)

Bottom colormap limit for HTO feature plots, defined as a percentile rank.

`--hto-max-percent` (float, default: 99)

Top colormap limit for HTO feature plots, defined as a percentile rank.

`--hto-min-value`, `--hto-max-value` (float)

Bottom and top colormap limits for HTO feature plots, defined as scalars. Note that scalars and percentile ranks cannot be used together in the same analysis.

Report

`--id` (string)

Sample ID to include in the output report.

`--description` (string)

Sample description to include in the output report.

Outputs

Fig. 10 is an overview of the structure and content of the output directory from PIPseeker full. This output can be used in subsequent PIPseeker workflows or with other tools, e.g., *Seurat*.

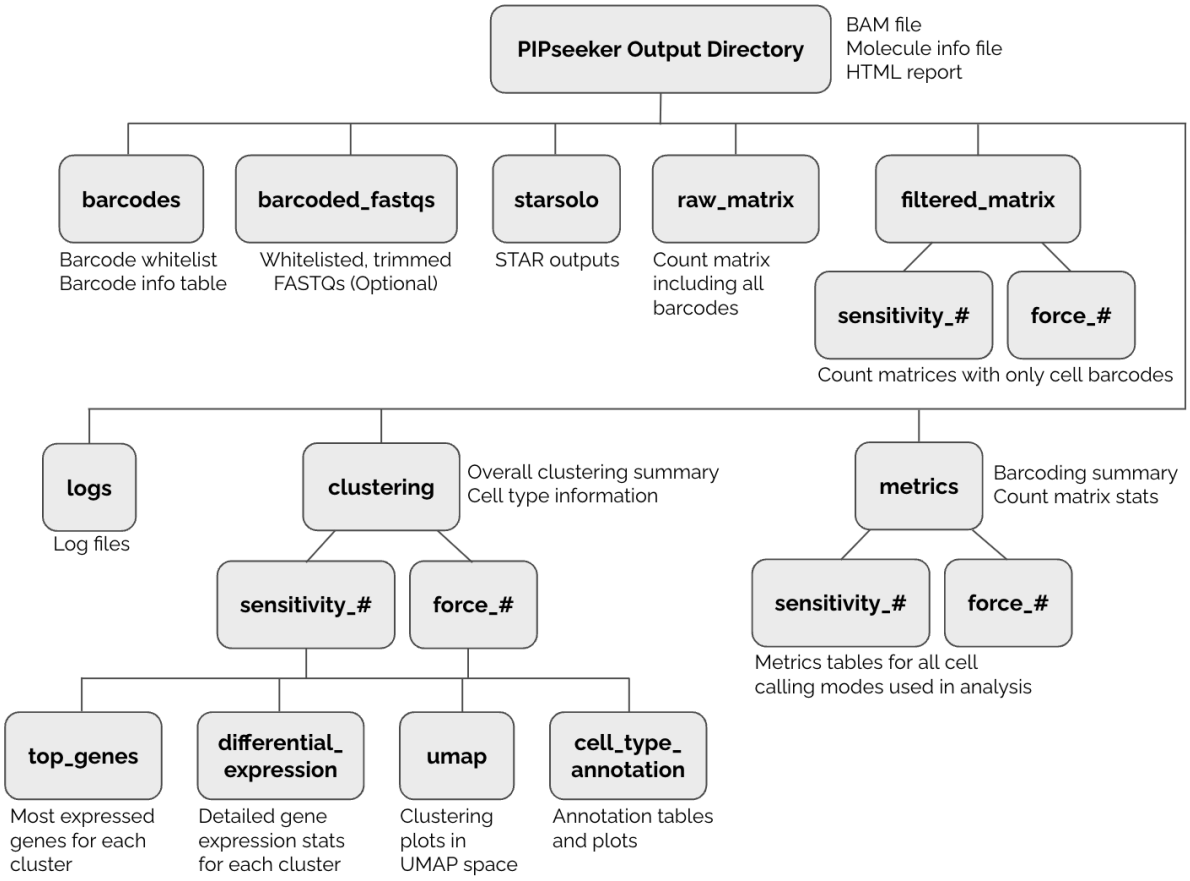

Figure 10: PIPseeker full outputs. Note that this diagram does not include specialized workflows, such as Barnyard, SNT and HTO.

Rerunning The Analysis Starting from Cell Calling: PIPseeker cells

After observing the results from PIPseeker full, you may want to change a parameter related to the late stages of the analysis, starting at cell calling, You may wish to force a certain number of barcodes to be designated as cells, or change other parameters such as the range of K values for K-means clustering or the number of differentially expressed genes to display. This does not require repeating

the entire workflow, which can be time consuming. Instead, the analysis can start directly with cell calling from the previously generated raw matrix.

PIPseeker cells takes as input the path to a previously generated PIPseeker result, as specified by the `--previous` argument. This will be the output directory from a prior PIPseeker full run. For example, assume that after running the full pipeline and visually inspecting the results, none of the standard cell-calling sensitivity levels seem to capture the correct region of the barcode rank plot. You can force a specific number of cells without re-running the entire pipeline:

```
$ <path to PIPseeker>/pipseeker cells --previous <PIPseeker full output
directory> --force-cells 5000
```

The results from the new analysis will be added to the directory previously used with PIPseeker full. A new tab representing the PIPseeker cells run (`force_5000` in the above example) will be added to the HTML report.

Input Arguments

All arguments described above for `PIPseeker full` are applicable in PIPseeker cells mode, except for those related specifically to the steps prior to building the raw count matrix.

`--previous` (string)

A path to a directory containing the result of a previously completed PIPseeker analysis. The raw matrix and other information will be loaded from this location.

`--hash` (string)

A hash label of an individual demultiplexed sample from cell hashing.

Reprocessing a Single Demultiplexed Sample

When using cell hashing to combine and demultiplex samples, it may be desirable to analyze different samples with different parameters. As part of the initial PIPseeker full run with HTOs, PIPseeker will have already processed each demultiplexed sample with the same parameters as were used to process the combined samples. Using PIPseeker cells with the `--hash` argument will start a reanalysis of only a single demultiplexed sample. For example:

```
$ <path to PIPseeker>/pipseeker cells --previous <PIPseeker full output
directory> --clustering-sensitivity high --min-clusters-kmeans 3
--max-clusters-kmeans 5 --hash <HTO name>
```

The command above will reprocess a specific demultiplexed sample with higher clustering sensitivity than the default, and will add K-means clustering in addition to graph-based clustering. Results will be output into the `HTO/demux` directory under the main results directory. Note that `--previous` should refer to the top results directory from a previous analysis, not a demultiplexed sample directory.

Generating FASTQ files with barcodes in standard format: PIPseeker barcode

As noted above, the PIPseq platform uses tiered barcodes, which are converted into contiguous 16-base barcodes in intermediate FASTQ files, that are then used for mapping. Some custom workflows may require those intermediate files, for example when running your own version of *STAR* or a custom aligner.

PIPseeker barcode generates FASTQ files and a barcode whitelist compatible with the standard format expected by most scRNA-seq processing tools. It stops after completing the barcoding step. The input parameters and the processing up to that point are identical to PIPseeker full.

```
$ <path to PIPseeker>/pipseeker barcode --fastq <FASTQ files prefix>
--chemistry <PIPseq chemistry version> --output-path <path to output directory>
```

Input Arguments

`--sorted-bam`

In addition to FASTQ files and a barcode whitelist, generate a BAM file in which reads are sorted by alignment position. This file is generated in addition to the conventional, unsorted BAM, and is not used by PIPseeker in the analysis process. It is meant to be used as input to other, downstream applications that require a sorted BAM.

Generating Gene Expression Feature Plots: PIPseeker feature

A common workflow following gene expression analysis is to plot expression levels for individual genes over the UMAP output of the clustering step. This can highlight differences in gene expression between cell populations. PIPseeker can generate any number of these plots based on previous analysis results.

PIPseeker feature generated UMAP plots of cell-normalized, log-transformed gene expression. It takes as input the path to a previously generated PIPseeker result, as specified by the `--previous` argument. It also requires a list of one or more genes, either provided directly in the command line or included in a file:

```
$ <path to PIPseeker>/pipseeker feature --previous <PIPseeker full output
directory> --genes Gene_1,Gene_2
```

or:

```
$ <path to PIPseeker>/pipseeker feature --previous <PIPseeker full output
directory> --genes-file genes.txt
```

Results will be output to a directory named `feature` in the directory defined by `--previous`. Plots will be saved in `plots`, and the corresponding data for each plot will be saved in `data`.

Fig. 11 shows examples of feature plots from two samples. Panel A shows cell type annotation and two genes in a human PBMC sample: MS4A1, which is typically expressed in B cells, and LYZ, a monocyte marker. Panel B shows a mouse brain sample, with plots for genes Prdx6 and Scn4b, markers for astrocytes and striatal D1 medium spiny neurons, respectively.

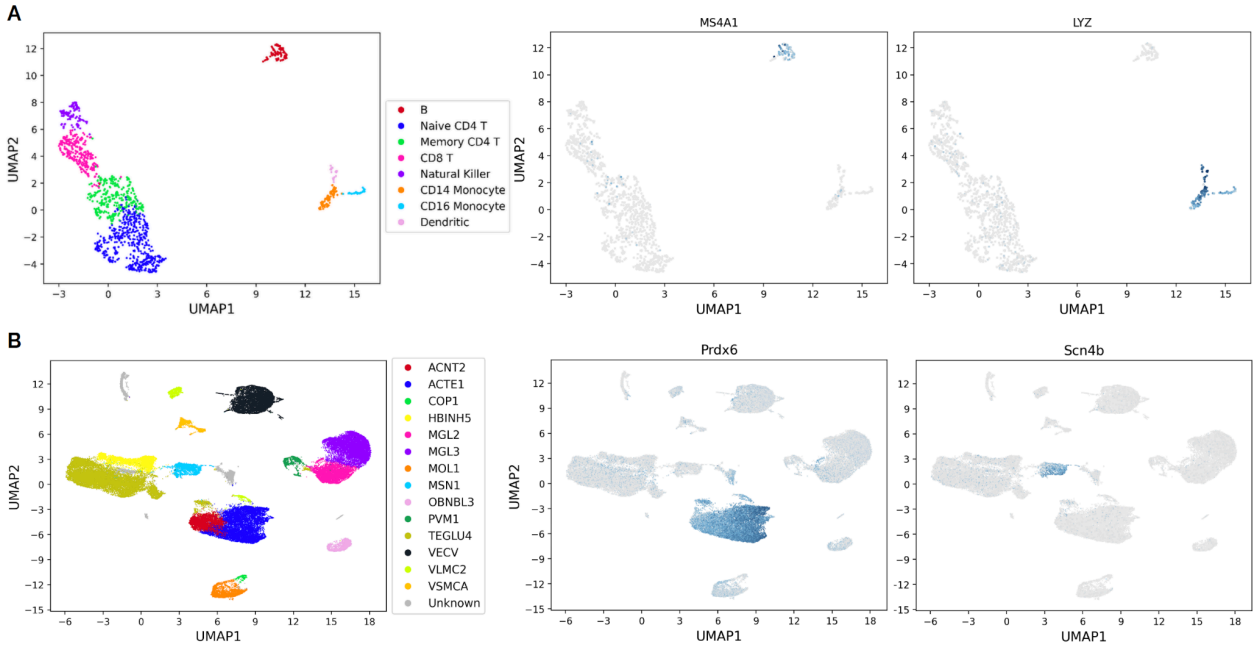

Figure 11: Feature plots from human PBMC and mouse brain samples. A: PBMC annotation and expression maps for MS4A1 and LYZ genes. B: Mouse brain annotation and expression maps for Prdx6 and Scn4b genes.

Input Arguments

The following arguments have the same functionality as in `full` or `cells` modes described above: `--previous`, `--skip-version-check`, `--verbosity`, `--umap-axes`, `--dpi`, `--save-svg`.

`--genes` (string)

A list of genes to create plots for. Should be a comma-separated list (no spaces), e.g., `--genes MALAT,MS4A1,CD14`

`--genes_file` (string)

A list of genes to create plots for. Should be a text file with one gene per row.

`--colormap` (string, default: gray-to-blue)

Colormap to use for plotting. See appendix 1 for available colormaps.

`--colorbar`

Add a colorbar to each feature plot.

`--min-percent` (float, default: 1)

Bottom colormap limit, defined as a percentile rank.

`--max-percent` (float, default: 99)

Top colormap limit, defined as a percentile rank.

`--min-value`, `--min-value` (float)

Bottom and top colormap limits, defined as scalars. Note that scalars and percentile ranks cannot be used together in the same analysis.

Generating Mapping References: PIPseeker buildmapref

Fluent BioSciences offers downloadable mapping references for several sample types, such as human, mouse and drosophila. Sometimes, however, other species are used, which require a custom mapping reference. PIPseeker buildmapref uses *STAR*'s genomeGenerate function to create a *STAR*-compatible reference, which can later be used for analysis.

PIPseeker buildmapref takes as input the path to one or more genome files in FASTA format and an annotation file in GTF format. The biotypes to include in the reference can be explicitly controlled. The output is a directory containing the suffix array and other reference files, which can be used as an input to PIPseeker full.

For example, the command below would produce a reference similar to the human reference on the Fluent BioSciences website. The FASTA and GTF files can be downloaded from GENCODE, ENSEMBL and other consortia.

```
$ <path to PIPseeker>/pipseeker buildmapref --fasta
GRCh38.primary_assembly.genome.fa --gtf
gencode.v39.primary_assembly.annotation.gtf --include-types
protein_coding,lncRNA,IG_C_gene,IG_D_gene,IG_J_gene,IG_LV_gene,IG_V_gene,TR_C_g
ene,TR_D_gene,TR_J_gene,TR_V_gene,IG_V_pseudogene,IG_J_pseudogene,IG_C_pseudoge
ne,TR_V_pseudogene,TR_J_pseudogene --output-path <output directory>
```

Similarly, the next example shows how to create the downloadable arabidopsis thaliana reference.

```
$ <path to PIPseeker>/pipseeker buildmapref --fasta TAIR10_GCA1735-1_genome.fa
--gtf TAIR10_GCA1735-1_annotation.gtf --exclude-types pseudogene --biotype-tag
gene_biotype --output-path <output directory>
```

Input Arguments

`--fasta` (string)

A path to one or more genome files in FASTA format. If specifying more than one file, use a comma-separated list (no spaces), e.g.: `file_1.fa,file_2.fa`

`--gtf` (string)

A path to a gene annotation file in GTF format. **Note:** Only the exon lines (lines with "exon" as the third field) are used by *STAR* to generate the reference. All other lines are ignored. Each line must include at a minimum a `gene_id` tag and a `transcript_id` tag. For example:

```
chr11   HAVANA  exon    65505204        65505278        .       +       .
gene_id "ENSG00000251562.9"; transcript_id "ENST00000617489.1"; gene_type
"lncRNA"; gene_name "MALAT1";
```

`--include-types`, `--exclude-types` (string)

Comma-separated list of biotypes to include or exclude from the reference. Only one of these arguments can be used. If neither is specified, all biotypes will be included. If biotypes are modified by the user, a filtered GTF file will be generated, saved to the output directory and used for creating the STAR index.

`--biotype-tag` (string, default: gene_type)

Tag in the GTF file to use for determining biotype. Only use in conjunction with `--include-biotypes` or `--exclude-biotypes` when explicitly specifying biotypes.

`--read-length` (integer, default: 100)

Expected length of read 2. This is used to adjust *STAR*'s `sjdbOverhang` parameter.

`--sparsity` (integer, default: 3)

Sparsity of suffix array. This is used to adjust *STAR*'s `genomeSAsparseD` parameter. Lower values will result in faster performance at the expense of bigger reference files and higher memory consumption.

`--additional-params` (string)

Additional STAR command-line parameters in the form: `--<name> <value>`. Input the entire set of parameter names and values as a single string enclosed in quotes.

`--threads` (integer, default: 0)

Number of threads to use for parallel processing (see full mode)

Generating Cell Type Annotation References: PIPseeker buildannotref

Fluent BioSciences offers downloadable annotation references for several sample types, such as human PBMC and mouse brain. Some users, however, may need annotations for alternative tissues or cell types. PIPseeker buildannotref uses the clustering results from one or more processed samples to create cell type annotation references that can be used to identify cells in future samples of the same type.

PIPseeker buildannotref takes as input the path to one or more previously generated PIPseeker results, as specified by the `--previous`, `--cell-calling-mode` and `--clustering-mode` arguments. It uses a CSV file with cell types manually assigned to clusters , as specified by `--clusters-file`, whereby it creates a specific signature for each cell type and builds an annotation reference. For example:

```
$ <path to PIPseeker>/pipseeker buildannotref --previous
my/analysis/result/dir/sample1,my/analysis/result/dir/sample2
--cell-calling-mode sensitivity_3,sensitivity_4 --clustering-mode graph,k_4
--clusters-file annotated_clusters.csv --output-file annotation_ref.csv
```

The generated annotation reference will include a list of differential expression scores for each cell type. The scores for cell types defined across multiple samples will be averaged between the samples. See Cell Type Annotation section above for more details about the classification algorithm.

Input Arguments

`--previous` (string)

A path to a directory containing the result of one or more previously completed PIPseeker analyses. If more than one previous result is used, this should be a comma-delimited list of paths.

`--cell-calling-mode` (string)

Cell calling mode (sensitivity_<N>, or force_<N> in case of manual cell calling). Should be a comma-delimited list if multiple samples are used.

`--clustering-mode` (string)

Clustering mode (graph, or k_<N> if K-means clustering was previously added). Should be a comma-delimited list if multiple samples are used.

`--clusters-file` (string)

A path to a CSV file with no header and the following columns:

1) Cluster ID, taken from the clustering results for each sample. The cluster numbers are listed in the output files and shown in the sample reports.

2) Cell type chosen for each cluster, e.g., Monocyte, Hepatocyte, CD34+ (May contain spaces).

3) If more than one sample is used, sample index (starting at 1), based on the order in which they are listed in `--previous`. This column can be omitted if only one sample is used.

Here is an example of a clusters file for PBMC annotation from two PBMC results:

```
4,B,1
1,Naive CD4 T,1
4,Naive CD4 T,2
10,CD8 T,1
11,Natural Killer,1
9,CD14 Monocyte,1
10,CD16 Monocyte,1
1,HSPC,2
```

Note that Naive CD4 is defined for both samples, which means the expression scores for this type will be averaged between the samples.

`--top-genes` (integer, default: 100)

The number of top differentially expressed genes to include from each cluster. The final set of genes included in the generated reference is the union of the lists of top expressing genes from each cluster.

`--weights` (list of integers or floats)

Weights to assign to samples if multiple previous results are included. Should be a comma-delimited list of numbers. This applies only to cell types defined in more than one sample, such as Naive CD4 in the example above. If not defined, all samples will be weighted equally.

`--output-file` (string)

Path to the output annotation reference file.

Merging PIPseeker Results: PIPseeker merge

Users may want to integrate multiple pre-existing PIPseeker results into a single dataset. This workflow may be useful when analyzing similar cell types or conditions across different experiments.

PIPseeker merge allows users to: (1) combine data from various batches or experiments into a single result, (2) correct batch effects if present, and (3) compare and contrast cellular behavior across multiple datasets.

PIPseeker merge (Fig. 12) takes as input the paths to two or more previously generated PIPseeker results (samples), as defined by the `--previous` and `--cell-calling-mode` arguments. A batch may be assigned to each sample using `--batches`. When `--batches` is provided, batch correction will be performed using the Harmony algorithm. Clustering and cell type annotation will be performed on the merged dataset.

Below are examples without batches and with batches, respectively:

```
$ <path to PIPseeker>/pipseeker merge --previous
my/analysis/result/dir/sample1,my/analysis/result/dir/sample2
--cell-calling-mode sensitivity_3,sensitivity_2 --output-path
/path/to/merged/output/dir --sample-labels "sample 1,sample 2" --annotation
/path/to/annotation_file.csv
```

or:

```
$ <path to PIPseeker>/pipseeker merge --previous
my/analysis/result/dir/sample1,my/analysis/result/dir/sample2,my/analysis/resul
t/dir/sample3,my/analysis/result/dir/sample4 --cell-calling-mode
sensitivity_3,sensitivity_2,sensitivity_3,sensitivity_4 --output-path
/path/to/merged/output/dir --batch control,control,treatment,treatment
```

PIPseeker merge will output cell calling, filtered matrix, and clustering results for the merged data, as well as a metadata file (metadata.csv) and an html report (report.html). All cells from each previous sample will be included in the called cells and filtered matrix. In addition to other clustering and cell type annotation results, output files include UMAPs colored by sample and batch labels, and a `merge_data` folder with cell counts from each sample and batch for each clustering mode. See Clustering and Cell Type Annotation sections above for more details about clustering, differential expression, and annotation outputs.

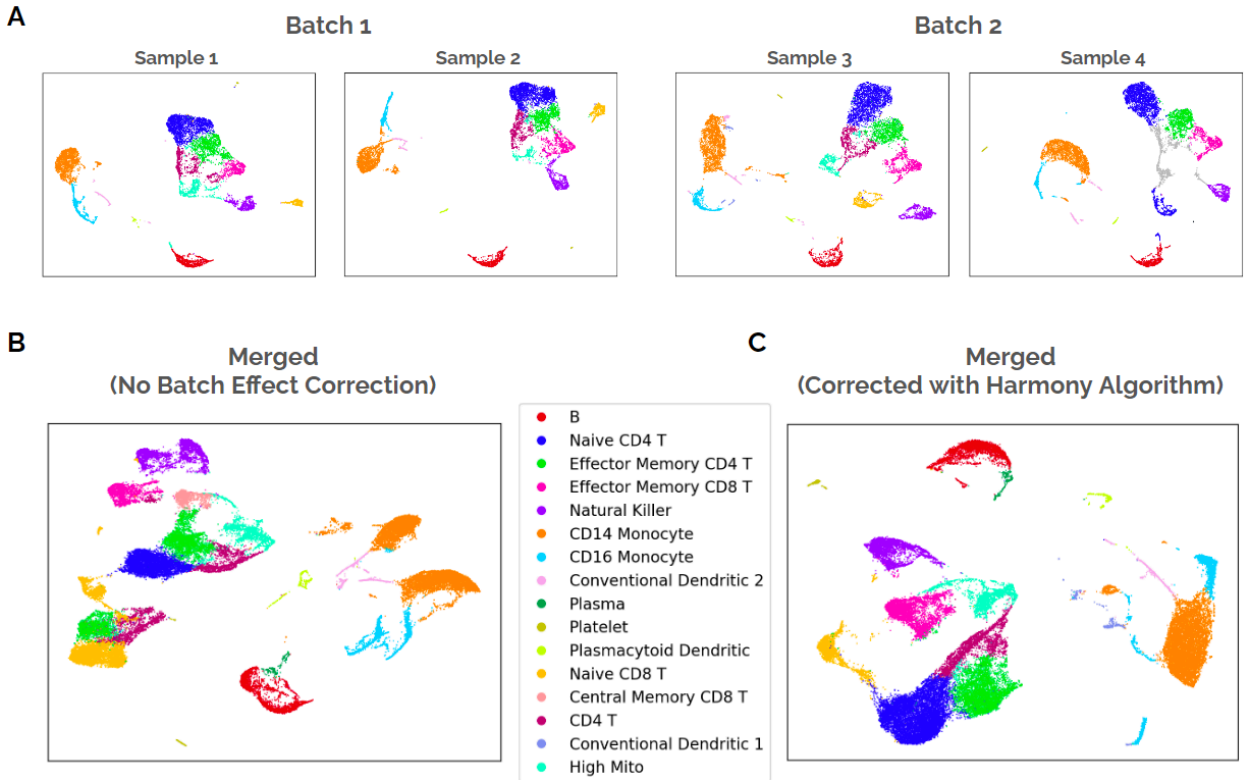

Figure 12: Combining human PBMC samples with PIPseeker merge. A: Annotated UMAPs for four human PBMC samples from two batches B: Annotated UMAP for samples from A merged with no correction. Note separation of clusters containing the same cell type from different batches. C: Annotated UMAP for samples from A merged with batch effect correction using Harmony.

Input Arguments

`--previous` (string)

A comma-separated list of paths to directories containing the results of previously completed PIPseeker analyses. May not include samples with SNT or HTO analysis.

`--cell-calling-mode` (string)

A comma-separated list of cell calling modes (sensitivity_<N> or force_<N>) the same length as --previous, containing the desired cell calling mode for each sample..

`--output-path` (string)

Path to the desired merged result output directory. If the directory does not exist, it will be created.

`--sample-labels` (string)

A comma-separated list of labels for each sample in --previous to use in output files. Must be the same length as --previous. By default, the base directory name for each path will be used as the sample label. If duplicate labels exist, 'sample_<N>' will be used instead.

`--batches` (string)

A comma-separated list of batch labels for each sample in --previous. If defined, batch effects will be corrected with Harmony, and batch outputs will be created. By default, no batches or batch effect correction will be applied.

`--harmony-principal-components` (integer)

An integer specifying the number of principal components to correct with harmony. Must be less than or equal to the number of principal components used in clustering (given by --principal-components). By default, the first 200 principal components are corrected.

`--harmony-k` (integer)

An integer specifying the number of clusters to use in the harmony correction model. This value should be close to the expected number of cell types present in the merged result. By default, the maximum number of graph-based clusters identified in the previous clustering results will be used.

`--harmony-block-size` (float, default: 0.05)

A positive value between 0 and 1 specifying the proportion of cells to be updated together in each block update during clustering. Higher values lead to faster runtime with less accurate results.

`--theta` (float, default: 2)

A non-negative value defining the penalty for batch diversity in clustering. Higher values encourage more batch diversity, splitting batches more evenly among clusters as theta increases, and giving no penalty for lack of diversity as theta approaches 0.

`--lam` (float, default: 1)

A positive value specifying the ridge regression penalty parameter used in the harmony correction steps. Smaller values give more aggressive correction.

`--sigma` (float, default: 0.1)

A positive value specifying the soft k-means clustering width parameter. As sigma approaches 0, k-means clustering results approach hard clustering. Higher values of sigma assign cells to more clusters.

`--tau` (float, default: 0)

A non-negative value specifying the expected number of cells per cluster. Larger values are used to prevent overclustering when batches of different sizes are present by penalizing creation of small clusters. If tau is 0, no penalty is applied

`--harmony-iter` (integer, default: 10)

A positive integer giving the maximum allowed iterations to run harmony. If convergence is not reached, an error is given.

`--clust-iter` (integer, default: 20)

A positive integer giving the maximum allowed iterations in each clustering round..

`--harmony-epsilon` (float, default: 1e-4)

A positive value setting the threshold for harmony convergence. Smaller values are a stricter convergence threshold.

`--clust-epsilon` (float, default: 1e-5)

A positive value setting the threshold for clustering convergence. Smaller values are a stricter convergence threshold.

Outputs

The file `called_cells.txt` in `<output path>/cell_calling/merge_<N>` contains the index of each cell corresponding to its barcode in `<output path>/filtered_matrix/merge_<N>`. Cell calling as described previously is not performed since all cells from previous results are combined in the merged dataset.

Merged clustering results in `<output path>/clustering/merge_<N>` contain the following in addition to clustering files described previously:

- `clusters.csv`: cluster assignment of each cell from graph-based clustering and from all the K-means runs. Includes sample label for each cell and batch label if applicable
- `umap/sample/*.png`: UMAP plots of sample labels. Includes `all_sample.png` and plots for each individual sample if three or more samples are merged. `umap/batch/*.png` also output if applicable.
- `merge_data/sample/*.csv`: Files containing count of cells from each sample present in each cluster for each clustering mode. `merge_data/batch/*.csv` also output if applicable.
- When cell type annotation is used:
    - `cell_type_annotation/<sample type>/merge_data/sample/*.csv`: Files containing count of cells from each sample present in each cell type for each clustering mode. `cell_type_annotation/<sample type>/merge_data/batch/*.csv` also output if applicable.

The directory `<output path>/filtered_matrix/merge_<N>` contains the merged counts matrix `matrix.mtx.gz` which combines the filtered matrix from each input previous result. The file `features.tsv.gz` is identical to the `features.tsv.gz` file in each previous directory. The `barcodes.tsv.gz` file contains the generated barcode associated with each cell in the merged

matrix. If cells from two previous results had the same barcode, a new barcode will be assigned to one of the cells in the merged dataset.

The file `metadata.csv` in `<output path>` contains the sample labels, batch labels (if applicable), cell calling modes, and previous directories used to generate the merged dataset in `<output path>`.

The file `report.html` in `<output path>` contains sample and batch (if applicable) merged figures and tables, as well as clustering, differential expression, and cell type annotation (if applicable) results.

Extracting Gene Expression Data: PIPseeker extract

By default, PIPseeker performs differential expression analysis for the top 50 differentially expressed genes between each cluster and all other clusters. However, users may want additional gene expression insight from their PIPseeker results. These insights may include: (1) expression of specific genes in certain clusters or (2) differential expression comparison between specific clusters or cell types. Additionally, users may want to extract gene expression insights from merged PIPseeker results, including: (1) expression of specific genes between batches or experiments or (2) expression of specific genes in a rare cell type that is only annotated when multiple samples are combined. PIPseeker extract performs these analyses using the clustering results from a previously processed sample. If the previous sample was processed with PIPseeker merge, expression results can be extracted from cells in specific samples or batches if desired.

PIPseeker extract takes as input the path to a previously generated PIPseeker result, as specified by the `--previous`, `--cell-calling-mode` and `--clustering-mode` arguments. A list of one or more genes is required as input, either provided directly in the command line or included in a text file. Users need to define two groups, indicated either as cluster numbers or, if cell type annotation was performed, as cell types. Based on user-specified cluster or cell type groups, PIPseeker calculates the mean expression of each input gene for each group and performs differential expression analysis between group 1 and group 2.

The clusters or cell types included in groups 1 and 2 can be provided as `--clusters-1` and `--clusters-2` or `--cell-types-1` and `--cell-types-2`. If a value or list of values are not explicitly provided for group 2, it will automatically default to all clusters or cell types not defined in group 1. If the `--exclude` flag is provided, those clusters or cell types will not be included in either group for the analysis.

If `--previous` contains merged results, `--sample-1` and `--sample-2` or `--batch-1` and `--batch-2` can additionally be used to specify which samples or batches the cells in each group should come from. In this case, both `--clusters-1` and `--clusters-2` or `--cell-types-1` and

`--cell-types-2` must be included, or if no analysis groups are defined, all cells from each sample or batch grouping will be included in each analysis group.

Below are examples using either clusters or cell types as input in combination with `--exclude`, `--genes`, and `--genes-file` flags, as well as sample and batch groupings:

Example 1: extract gene expression stats for Gene_1 and Gene_2 from the specified previous sample using sensitivity 3 graph-based clustering results. Group 1 includes all cells in graph-based cluster 1, and group 2 contains all remaining cells. Output files have the prefix 'analysis1'.

```
$ <path to PIPseeker>/pipseeker extract --previous
my/analysis/result/dir/sample --cell-calling-mode sensitivity_3
--clustering-mode graph --clusters-1 1 --genes Gene_1,Gene_2 --file-prefix
analysis1
```

Example 2: extract gene expression stats for Gene_1 and Gene_2 from the merged previous results using merge_1000 graph-based clustering results . Group 1 includes cells in graph-based clusters 1 and 2 from sample 'sample_1', and group 2 contains cells in graph-based cluster 3 from sample 'sample_2'.

```
$ <path to PIPseeker>/pipseeker extract --previous
my/analysis/result/dir/sample --cell-calling-mode merge_1000 --clusters-1 1,2
--clusters-2 3 --genes Gene_1,Gene_2 --sample-1 sample_1 --sample-2 sample_1
```

Example 3: extract gene expression stats for each gene in genes.txt from the specified previous sample using sensitivity 3 graph-based clustering cell type annotation results. Group 1 includes all CD8 T and Naive CD4 T cells, and group 2 contains all B cells.

```
$ <path to PIPseeker>/pipseeker extract --previous
my/analysis/result/dir/sample --cell-calling-mode sensitivity_3
--clustering-mode graph --cell-types-1 "CD8 T,Naive CD4 T" --cell-types-2 B
--genes-file genes.txt
```

Example 4: extract gene expression stats for Gene_1 from the specified previous sample using sensitivity 3 k_5 k-means clustering results. Group 1 includes all cells in clusters 1 and 2, and group 2 contains all cells in cluster 4.

```
$ <path to PIPseeker>/pipseeker extract --previous
my/analysis/result/dir/sample --cell-calling-mode sensitivity_3
--clustering-mode k_5 --clusters-1 1,2 --exclude 3,5 --genes Gene_1
```

Example 5: extract gene expression stats for Gene_1 and Gene_2 from the merged previous results using merge_5000 graph-based clustering cell type annotation results. Group 1 includes B cells in batch 'batch_1', and group 2 contains B cells in batch 'batch_2'.

```
$ <path to PIPseeker>/pipseeker extract --previous
my/analysis/result/dir/sample --cell-calling-mode merge_5000 --cell-types-1 B
--cell-types-2 B --genes Gene_1,Gene_2 --batch-1 batch_1 --batch-2 batch_2
```

Example 6: extract gene expression stats for each gene in genes.txt from the merged previous results using merge_5000 cell calling mode. Group 1 includes all cells in batch 'batch_1', and group 2 contains all cells in batch 'batch_2'.

```
$ <path to PIPseeker>/pipseeker extract --previous
my/analysis/result/dir/sample --cell-calling-mode merge_5000 --genes-file
genes.txt --batch-1 batch_1 --batch-2 batch_2
```

PIPseeker extract will output a generated stats file (stats.csv) that includes mean and standard deviation of expression in group 1 and group 2 for each gene, as well as differential expression scores. Information about clusters or cell types in each analysis group, samples or batches in each analysis group (if applicable), and the annotation reference used (if applicable) will be included in a metadata file `metadata.txt`. Optional barcharts (<gene>_barchart.png) with mean expression of each gene can also be output by including the `--barcharts` argument. See Clustering and Cell Type Annotation sections above for more details about differential expression scores and annotation outputs.

Input Arguments

`--previous` (string)

A path to a directory containing the result of a previously completed PIPseeker analysis.

`--cell-calling-mode` (string)

Cell calling mode (sensitivity_<N> or force_<N>).

`--clustering-mode` (string)

Clustering mode (graph or k_<N> if K-means clustering was previously performed). Default is graph.

`--genes` (string)

A gene or list of genes to use for expression analysis. Should be a comma-separated list (no spaces), e.g., `--genes MALAT,MS4A1,CD14`

`--genes-file` (string)

A file containing a list of genes to use for expression analysis. Should be a text file with one gene per row.

`--clusters-1` (integer or list of integers)

A cluster or list of clusters to use for analysis group 1. Should be a comma-separated list (no spaces), e.g., `--clusters-1 1,2,3`

`--clusters-2` (integer or list of integers)

A cluster or list of clusters to use for analysis group 2. Should be a comma-separated list, e.g., `--clusters-2 4,5.` If not specified, group 2 defaults to all clusters not in `--clusters-1`.

`--cell-types-1` (string)

A cell type or list of cell types to use for analysis group 1. Should be a comma-separated list (no spaces). If any of the cell types contain spaces, the list should be enclosed in quotes, e.g., `--cell-types-1 "B,CD8 T,HSPC".`

`--cell-types-2` (string)

A cell type or list of cell types to use for analysis group 2. Should be a comma-separated list,, e.g., `--cell-types-2 "Naive CD4 T,CD16 Monocyte".` If not specified, group 2 defaults to all cell types not in `--cell-types-1`.

`--exclude` (string)

A cell type, cluster, or list of cell types or clusters to exclude from analysis group 2. Should be a comma-separated list, e.g., `--exclude "Natural Killer" or --exclude 7,8.` Applicable only if `--clusters-2 and --cell-types-2` are not specified (cannot be used with batch or sample groups). Group 2 will become all values not included in group 1 or excluded by `--exclude`.

`--sample-1` (string)

A sample or list of samples to select analysis group 1 cells from. Should be a comma-separated list (no spaces). If any of the sample labels contain spaces, the list should be enclosed in quotes, e.g., `--sample-1 "sample 1, sample 2".` If sample groups are used, --previous must contain merged results from PIPseeker merge. Both --sample-1 and --sample-2 must be defined, as well as cell type or cluster analysis group 1 and 2. If neither group 1 or group 2 is defined, --sample-1 and --sample-2 must contain non-overlapping samples, and all cells will be included in each analysis group.

`--sample-2` (string)

A sample or list of samples to select analysis group 2 cells from. Should be a comma-separated list (no spaces). If any of the sample labels contain spaces, the list should be enclosed in quotes, e.g., `--sample-2 "sample 1, sample 2".`

`--batch-1` (string)

A batch or list of batches to select analysis group 1 cells from. Should be a comma-separated list (no spaces). If any of the batch labels contain spaces, the list should be enclosed in quotes, e.g., `--batch-1 "batch 1, batch 2".` If batch groups are used, --previous must contain merged and batch corrected results from PIPseeker merge. Both --batch-1 and --batch-2 must be defined, as well as cell type or cluster analysis group 1 and 2. If neither group 1 or group 2 is defined, --batch-1 and --batch-2 must contain non-overlapping batches, and all cells will be included in each analysis group.

`--batch-2` (string)

A batch or list of batches to select analysis group 2 cells from. Should be a comma-separated list (no spaces). If any of the batch labels contain spaces, the list should be enclosed in quotes, e.g., `--batch "batch 1, batch 2".`

`--barcharts`

Include mean expression barcharts as output for each gene.

`--annotation-ref` (string)

Required only when the previous PIPseeker results directory includes cell type annotation results from multiple references. Identifies the name of the annotation reference directory in the cell_type_annotation subfolder of the clustering directory.

`--file-prefix` (string)

Prefix to use for output files, including stats and metadata files. For example, if set to `prefix1`, the output files will be prefix1_stats.csv and prefix1_metadata.txt. By default, no prefix will be used.

Appendix 1: Colormaps

PIPseeker can create feature UMAP plots for genes and synthetic tags using a variety of colormaps. The recommended colormaps are shown below.

Using a colormap that begins with white will result in non-expressing cells being invisible against the white background. To make them visible, choose a colormap that begins with gray.

In addition, any of the standard colormaps listed here can be used:
https://matplotlib.org/stable/gallery/color/colormap_reference.html

Document Revision Summary

Doc ID: FB0002787
Revision: 8
Revision Date: June 2024

General Changes:

- PIPseq V molecular counting
- Input argument changes

Legal Notices

Support

Email: support@fluentbio.com

Fluent BioSciences
150 Coolidge Avenue
Watertown, MA 02472