VSN-Pipelines

Apr 19, 2023

Contents

CHAPTER 1

Prerequisite

Make sure that `LANG` and `LC_ALL` environment variables have been set. You can use the following command to check this:

```
locale
```

If some are not set, you can set them to the default language for instance:

```
export LANG="C"
export LC_ALL="C"
```

1.1 Dependencies

Make sure you have the following software installed,

- Nextflow
  - Currently VSN-Pipelines requires Nextflow version `21.04.03` or higher.
- A container system, either of:
  - Docker
  - Singularity

NOTE: Due to licensing restrictions, to use the cellranger components of VSN you must build and/or provide a container with `cellranger` and `bcl2fastq2` installed yourself. A sample `Dockerfile` can be found in `./src/cellranger/`, you must download `bcl2fastq2` from the Illumina website, and cellranger from the 10x Genomics website yourself to build this container.

CHAPTER 2

Quick start

To run a quick test of the single sample analysis pipeline, we can use the 1k PBMC datasets provided by 10x Genomics. This will take only **~3min** to run.

1. The data first needs to be downloaded (instructions can be found here).
2. Next, update to the latest pipeline version:

```
nextflow pull vib-singlecell-nf/vsn-pipelines
```

3. Next, generate a config file using the standard settings for the test data, and the appropriate profiles (e.g., replace `singularity` with `docker` if necessary):

```
nextflow config vib-singlecell-nf/vsn-pipelines \
    -profile tenx,singularity,single_sample > single_sample.config
```

4. The test pipeline can now be run using the config file just generated, specifying the `single_sample` workflow as an entrypoint:

```
nextflow -C single_sample.config \
    run vib-singlecell-nf/vsn-pipelines \
        -entry single_sample
```

2.1 Example Output

```
$ nextflow -C nextflow_new_structure.config run $VSN -entry single_sample
N E X T F L O W  ~  version 21.04.3
Launching `/staging/leuven/stg_00002/lcb/dwmax/documents/aertslab/GitHub/vib-
↪singlecell-nf/vsn-pipelines/main.nf` [loving_shockley] - revision: ba1dedbf51
executor >  local (75)
[1d/3b5a55] process > single_sample:SINGLE_SAMPLE:SC__FILE_CONVERTER (2)
↪                                                                   [100%] 2 of 2 _
[2d/2152cf] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:QC_FILTER:SC__
↪SCANPY__COMPUTE_QC_STATS (2)                                       [100%] 2 of 2 _
```

(continues on next page)

(continued from previous page)

```
[48/bce024] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:QC_FILTER:SC__
↪SCANPY__CELL_FILTER (2)                                    [100%] 2 of 2 _
[60/d42cdf] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:QC_FILTER:SC__
↪SCANPY__GENE_FILTER (2)                                    [100%] 2 of 2 _
[4b/bb2635] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:QC_
↪FILTER:GENERATE_DUAL_INPUT_REPORT:SC__SCANPY__GENERATE_DUAL_INPUT_REPORT (2) [100%]␣
↪2 of 2 _
[64/add548] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:QC_
↪FILTER:GENERATE_DUAL_INPUT_REPORT:SC__SCANPY__REPORT_TO_HTML (2)           [100%]␣
↪2 of 2 _
[c6/4d8a66] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:NORMALIZE_
↪TRANSFORM:SC__SCANPY__NORMALIZATION (2)                    [100%] 2 of 2␣
↪_
[01/8ba0d2] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:NORMALIZE_
↪TRANSFORM:PUBLISH_H5AD_NORMALIZED:COMPRESS_HDF5 (2)        [100%] 2 of 2␣
↪_
[b3/ec4712] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:NORMALIZE_
↪TRANSFORM:PUBLISH_H5AD_NORMALIZED:SC__PUBLISH (2)          [100%] 2 of 2␣
↪_
[1e/35bb2e] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:NORMALIZE_
↪TRANSFORM:SC__SCANPY__DATA_TRANSFORMATION (2)              [100%] 2 of 2␣
↪_
[14/adfd7a] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:HVG_
↪SELECTION:SC__SCANPY__FIND_HIGHLY_VARIABLE_GENES (2)                [100%]␣
↪2 of 2 _
[01/9c8a26] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:HVG_
↪SELECTION:SC__SCANPY__SUBSET_HIGHLY_VARIABLE_GENES (2)              [100%]␣
↪2 of 2 _
[dc/027334] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:HVG_
↪SELECTION:SC__SCANPY__FEATURE_SCALING (2)                           [100%]␣
↪2 of 2 _
[8d/05ce2f] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:HVG_
↪SELECTION:PUBLISH_H5AD_HVG_SCALED:COMPRESS_HDF5 (2)                 [100%]␣
↪2 of 2 _
[0b/6d50b0] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:HVG_
↪SELECTION:PUBLISH_H5AD_HVG_SCALED:SC__PUBLISH (2)                   [100%]␣
↪2 of 2 _
[c1/f799be] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:HVG_
↪SELECTION:GENERATE_REPORT:SC__SCANPY__GENERATE_REPORT (2)           [100%]␣
↪2 of 2 _
[c9/ae0cd9] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:HVG_
↪SELECTION:GENERATE_REPORT:SC__SCANPY__REPORT_TO_HTML (2)            [100%]␣
↪2 of 2 _
[a2/0a7824] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:DIM_REDUCTION_
↪PCA:SC__SCANPY__DIM_REDUCTION__PCA (2)                     [100%] 2 of 2 _
[d6/1407b3] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:NEIGHBORHOOD_
↪GRAPH:SC__SCANPY__NEIGHBORHOOD_GRAPH (2)                    [100%] 2 of 2 _
[b7/8ab962] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:DIM_REDUCTION_
↪TSNE_UMAP:SC__SCANPY__DIM_REDUCTION__TSNE (2)              [100%] 2 of 2 _
[ee/485413] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:DIM_REDUCTION_
↪TSNE_UMAP:SC__SCANPY__DIM_REDUCTION__UMAP (2)              [100%] 2 of 2 _
[ba/2bfd23] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:DIM_REDUCTION_
↪TSNE_UMAP:GENERATE_REPORT:SC__SCANPY__GENERATE_REPORT (2)   [100%] 2 of 2 _
[10/a429ce] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:DIM_REDUCTION_
↪TSNE_UMAP:GENERATE_REPORT:SC__SCANPY__REPORT_TO_HTML (2)    [100%] 2 of 2 _
[06/3412cd] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:CLUSTER_
↪IDENTIFICATION:SC__SCANPY__CLUSTERING (2)                          [100%] 2 of␣
↪2 _
```

(continues on next page)

(continued from previous page)

```
[23/3d82c4] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:CLUSTER_
↪IDENTIFICATION:GENERATE_REPORT:SC__SCANPY__GENERATE_REPORT (2)          [100%] 2 of␣
↪2 _
[bb/c9e11f] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:CLUSTER_
↪IDENTIFICATION:GENERATE_REPORT:SC__SCANPY__REPORT_TO_HTML (2)           [100%] 2 of␣
↪2 _
[1c/2026be] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:CLUSTER_
↪IDENTIFICATION:SC__SCANPY__MARKER_GENES (2)                             [100%] 2 of␣
↪2 _
[57/13f0a8] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:UTILS__
↪GENERATE_WORKFLOW_CONFIG_REPORT                                          [100%] 1␣
↪of 1 _
[60/8a3231] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:SC__SCANPY__
↪MERGE_REPORTS (2)                                                     [100%] 2 of 2 _
[cb/de1a4d] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:SC__SCANPY__
↪REPORT_TO_HTML (2)                                                    [100%] 2 of 2 _
[3f/265503] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:FINALIZE:SC__
↪H5AD_TO_FILTERED_LOOM (2)                                             [100%] 2 of 2 _
[1f/de67e8] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:FINALIZE:FILE_
↪CONVERTER_TO_SCOPE:SC__H5AD_TO_LOOM (2)                               [100%] 2 of 2 _
[2a/10d5a2] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:FINALIZE:FILE_
↪CONVERTER_TO_SCANPY:SC__H5AD_MERGE (2)                                [100%] 2 of 2 _
[35/ce7256] process > single_sample:SINGLE_SAMPLE:SCANPY__SINGLE_SAMPLE:PUBLISH:SC__
↪PUBLISH (2)                                                           [100%] 2 of 2 _
[6f/1df294] process > single_sample:PUBLISH_SINGLE_SAMPLE_SCOPE:COMPRESS_HDF5 (2)     ␣
↪                                                                      [100%] 2 of 2 _
[90/a9e563] process > single_sample:PUBLISH_SINGLE_SAMPLE_SCOPE:SC__PUBLISH (2)       ␣
↪                                                                      [100%] 2 of 2 _
[23/d62b2e] process > single_sample:PUBLISH_SINGLE_SAMPLE_SCANPY:COMPRESS_HDF5 (2)    ␣
↪                                                                      [100%] 2 of 2 _
[d0/5834be] process > single_sample:PUBLISH_SINGLE_SAMPLE_SCANPY:SC__PUBLISH (1)      ␣
↪                                                                      [100%] 2 of 2 _
WARN: To render the execution DAG in the required format it is required to install␣
↪Graphviz -- See http://www.graphviz.org for more info.
Completed at: 26-Aug-2021 15:41:37
Duration    : 2m 29s
CPU hours   : 0.4
Succeeded   : 75
```

2.2 Output

The pipelines will generate 3 types of results in the output directory (*params.global.outdir*), by default `out/`

- `data`: contains the workflow output file (in h5ad format), plus symlinks to all the intermediate files.
- `loom`: contains final loom files which can be imported inside SCope visualization tool for further visualization of the results.
- `notebooks`: contains all the notebooks generated along the pipeline (e.g.: Quality control report)
- `pipeline_reports`: Nextflow dag, execution, timeline, and trace reports

For a full list of the pipelines available please see the pipelines page.

CHAPTER 3

Further pipeline configuration details

This pipeline can be fully configured and run on custom data with a few steps. The recommended method is to first run `nextflow config ...` to generate a complete config file (with the default parameters) in your working directory. The tool-specific parameters, as well as Docker/Singularity profiles, are included when specifying the appropriate profiles to `nextflow config`.

1. First, update to the latest pipeline version (this will update the Nextflow cache of the repository, typically located in `~/.nextflow/assets/vib-singlecell-nf/`):

```
nextflow pull vib-singlecell-nf/vsn-pipelines
```

2. Next, a config file needs to be generated. This step will merge parameters from multiple profiles together to create a master config which specifies **all** parameters used by the pipeline. In this example, these are `tenx` for the input data, `singularity` to use the Singularity system (replace with `docker` if necessary), and `single_sample` to load the defaults for the single sample pipeline. In your working directory, run `nextflow config ...` with the appropriate profiles:

```
nextflow config vib-singlecell-nf/vsn-pipelines \
    -profile tenx,singularity,single_sample > single_sample.config
```

3. Now, edits can be made to `single_sample.config`. Generally, the default values are acceptable to use for a first pass, but certain variables (input directory, etc.) need to be changed.

   In particular, the following parameters are frequently modified in practice:

   - `params.global.project_name`: a project name which will be included in some of the output file names.
   - `params.data.tenx.cellranger_mex`, which should point to the `outs/` folder generated by Cell Ranger (if using 10x data). See *Information on using 10x Genomics datasets* for additional info.
   - Filtering parameters (`params.tools.scanpy.filter`): filtering parameters, which will be applied to all samples, can be set here: min/max genes, mitochondrial read fraction, and min cells. See *Multi-sample parameters* for additional info on how to specify sample-specific parameters.
   - Louvain cluster resolution: `params.tools.scanpy.clustering.resolution`.

- *Cell-* and *sample-* level annotations are also possible.

4. Run the workflow using the new config file (using `-C` is recommended to use **only** this file), specifying the proper workflow as the entry point:

```
nextflow -C single_sample.config \
    run vib-singlecell-nf/vsn-pipelines \
    -entry single_sample
```

3.1 Additional resources for running on custom data

- Input file formats available.
- Available pipelines.
- Advanced features for customizing pipelines.

Finally, see the list of case studies with specific examples and full config files at VSN-Pipelines-examples.

CHAPTER 4

Input Data Formats

Depending on the type of data you run the pipeline with, one or more appropriate profiles should be set when running `nextflow config`. These profiles are indicated in the sections below.

4.1 Specifying multiple samples

All the input data parameters are compatible with the following features:

- Glob patterns

```
"data/10x/1k_pbmc/1k_pbmc_*/outs/"
```

- Comma separated paths (paths can contain glob patterns)

```
"data/10x/1k_pbmc/1k_pbmc_v2_chemistry/outs/, data/10x/1k_pbmc/1k_pbmc_v3_chemistry/
↪outs/"
```

- Array of paths (paths can contain glob patterns)

```
[
    "data/10x/1k_pbmc/1k_pbmc_v2_chemistry/outs/",
    "data/10x/1k_pbmc/1k_pbmc_v3_chemistry/outs/"
]
```

4.2 Cell Ranger (10x Genomics)

Data from a standard Cell Ranger output directory can be easily ingested into the pipeline by using the proper input channel (`tenx_mex` or `tenx_h5`, depending on which file should be used). Multiple samples can be selected by providing the path to this directory using glob patterns.

```
/home/data/
└── cellranger
    ├── sample_A
    │   └── outs
    │       ├── filtered_feature_bc_matrix
    │       │   ├── barcodes.tsv
    │       │   ├── genes.tsv
    │       │   └── matrix.mtx
    │       └── filtered_feature_bc_matrix.h5
    └── sample_B
        └── outs
            ├── filtered_feature_bc_matrix
            │   ├── barcodes.tsv
            │   ├── genes.tsv
            │   └── matrix.mtx
            └── filtered_feature_bc_matrix.h5
```

4.2.1 MEX

To use the Cell Ranger Market Exchange (**MEX**) files, use the following profile when generating the config file:

```
-profile tenx
```

This profile adds the following parameter (`params.data.tenx.cellranger_mex`) into the generated .config file:

```
[...]
data {
    tenx {
        cellranger_mex = "/home/data/cellranger/sample*/outs/"
    }
}
[...]
```

4.2.2 H5

To use the Cell Ranger `h5` file as input, use the following profile:

```
-profile tenx_h5
```

This profile adds the `params.data.tenx.cellranger_h5` parameter into the generated .config file:

```
[...]
data {
    tenx {
        cellranger_h5 = "/home/data/cellranger/sample*/outs/"
    }
}
[...]
```

4.2.3 Input file detection

Setting the input directory appropriately, using a glob in the directory path in place of the sample names, will collect all the samples listed in the `filtered_[feature|gene]_bc_matrix` directories listed above. For example, in `params.data.tenx`, setting:

```
cellranger_mex = "/home/data/cellranger/sample*/outs/"
```

or

```
cellranger_h5 = "/home/data/cellranger/sample*/outs/"
```

will recursively find all 10x samples in that directory.

The pipeline will use either the `outs/filtered_feature_bc_matrix/` or the `outs/raw_feature_bc_matrix/` depending on the setting of the `params.utils.file_converter.useFilteredMatrix` (`true` uses filtered; `false` uses raw).

4.3 H5AD (Scanpy)

Use the following profile when generating the config file:

```
-profile h5ad
```

In the generated .config file, make sure the `file_paths` parameter is set with the paths to the `.h5ad` files:

```
[...]
data {
    h5ad {
        file_paths = "data/1k_pbmc_v*_chemistry_SUFFIX.SC__FILE_CONVERTER.h5ad"
        suffix = "_SUFFIX.SC__FILE_CONVERTER.h5ad"
    }
}
[...]
```

- The `suffix` parameter is used to infer the sample name from the file paths (it is removed from the input file path to derive a sample name).

In case there are multiple .h5ad files that need to be processed with different suffixes, the multi-labelled strategy should be used to define the h5ad parameter:

```
[...]
data {
    h5ad {
        GROUP1 {
            file_paths = "[path-to-group1-files]/*.SUFFIX1.h5ad"
            suffix = ".SUFFIX1.h5ad"
            group = ["technology", "10x"]
        }
        GROUP2 {
            file_paths = "[path-to-group1-files]/*.SUFFIX2.h5ad"
            suffix = ".SUFFIX2.h5ad"
            group = ["technology", "smart-seq2"]
        }
```

(continues on next page)

(continued from previous page)

```
    }
}
[...]
```

Notes:

- `GROUP1`, `GROUP2` are just example names here. They can be replaced by any value as long as they are alphanumeric (underscores are allowed).
- All the different *suffix* defined should unique.
- `file_paths` and `suffix` do allow list of paths/globs in the multi-labelled strategy.
- `group` [optional] should be an array of 2 elements where first element define the group name and the second the group value. This will add cell-based annotation for each group of files

---

4.4 Loom

Use the following profile when generating the config file:

```
-profile loom
```

In the generated .config file, make sure the `file_paths` parameter is set with the paths to the `.loom` files:

```
[...]
data {
    loom {
        file_paths = "data/1k_pbmc_v*_chemistry_SUFFIX.SC__FILE_CONVERTER.loom"
        suffix = "_SUFFIX.SC__FILE_CONVERTER.loom"
    }
}
[...]
```

- The `suffix` parameter is used to infer the sample name from the file paths (it is removed from the input file path to derive a sample name).

4.5 Seurat Rds

Use the following profile when generating the config file:

```
-profile seurat_rds
```

In the generated .config file, make sure the `file_paths` parameter is set with the paths to the `.Rds` files:

```
[...]
data {
    seurat_rds {
        file_paths = "data/1k_pbmc_v*_chemistry_SUFFIX.SC__FILE_CONVERTER.Rds"
        suffix = "_SUFFIX.SC__FILE_CONVERTER.Rds"
```

(continues on next page)

(continued from previous page)

```
    }
}
[...]
```

- The pipelines expect a Seurat v3 object contained in the .Rds file. (Seurat v2 objects are currently not supported).
- The `suffix` parameter is used to infer the sample name from the file paths (it is removed from the input file path to derive a sample name).

4.6 TSV

Use the following profile when generating the config file:

```
-profile tsv
```

In the generated .config file, make sure the `file_paths` parameter is set with the paths to the `.tsv` files:

```
[...]
data {
    h5ad {
        file_paths = "data/1k_pbmc_v*_chemistry_SUFFIX.SC__FILE_CONVERTER.tsv"
        suffix = "_SUFFIX.SC__FILE_CONVERTER.tsv"
    }
}
[...]
```

- The `suffix` parameter is used to infer the sample name from the file paths (it is removed from the input file path to derive a sample name).

4.7 CSV

Use the following profile when generating the config file:

```
-profile csv
```

In the generated .config file, make sure the `file_paths` parameter is set with the paths to the `.csv` files:

```
[...]
data {
    h5ad {
        file_paths = "data/1k_pbmc_v*_chemistry_SUFFIX.SC__FILE_CONVERTER.csv"
        suffix = "_SUFFIX.SC__FILE_CONVERTER.csv"
    }
}
[...]
```

- The `suffix` parameter is used to infer the sample name from the file paths (it is removed from the input file path to derive a sample name).

CHAPTER 5

Pipelines

5.1 Single-sample Pipelines

Pipelines to run on a single sample or multiple samples separately and in parallel.

5.1.1 single_sample

The `single_sample` workflow will process 10x data, taking in 10x-structured data, and metadata file. The standard analysis steps are run: filtering, normalization, log-transformation, HVG selection, dimensionality reduction, clustering, and loom file generation. The output is a loom file with the results embedded.

5.1.2 single_sample_scenic

Runs the `single_sample` workflow above, then runs the `scenic` workflow on the output, generating a comprehensive loom file with the combined results. This could be very resource intensive, depending on the dataset.

5.1.3 single_sample_scrublet

Runs the `single_sample` workflow above together with the Scrublet workflow.

The `single_sample` workflow is running from the input data. The `scrublet` workflow is running from the input data. The final processed file from the `single_sample` pipeline is annotated with the cell-based data generated by Scrublet.

The pipelines generate the following relevant files for each sample:

Table 1: Output Files (not exhaustive list)

Output File	Description
out/data/.SINGLE_SAMPLE_SCRUBLET.loom*	Same as *out/data/*.SCRUBLET.loom* but containing resulting loom file from a *single_sample* workflow but with additional metadata (doublet scores and predicted doublet for the cells) based on Scrublet run.
out/data/scrublet/.SCRUBLET_DOUBLETS_DETECTION.ScrubletObject.pklz*	Pickled file containing the Scrublet object.
out/data/scrublet/.SCRUBLET_ANNOTATED_BY_CELL_METADATA.h5ad*	H5AD file with raw data and doublets annotated
out/data/scrublet/.SINGLE_SAMPLE_SCRUBLET.h5ad*	Same as *out/data/*.SCRUBLET.h5ad* resulting from a `single_sample` workflow run and with doublets (inferred from Scrublet) removed.

Cuurently there are 3 methods available to call doublets from Scrublet doublet scores:

1. (Default) Scrublet will try to automatically identify the doublet score threshold. The threshold is then used to call doublets based on the doublet scores available in the *scrublet__doublet_scores* column. The doublets called are available in the *scrublet__predicted_doublets* column.
2. It can happen that Scrublet fails to find the automatic treshold. In that case, the pipeline will fail and let you know that either the method define in 3. has to be used or a custom threshold has to be provided. Either way, the pipeline will generate the Scrublet histograms. This is helpful especially if the user decide to select a custom threshold which will need to be reflected in the config as follows:

```
params {
    tools {
        scublet {
            threshold = [
              "<sample-name>": <custom-threshold>
            ]
        }
    }
}
```

3. This method is specifc to sample generated by the 10x Genomics single-cell platform. This method is based on the rate of the expected number of doublets in 10x Genomics samples. The number of doublets called (D) will be equal to the rate of doublets (given a number of cells) times the number of cells in that 10x Genomics sample. The cells are then ranked by their Scrublet doublet score (descending order) and the top D cells are called as doublets.

5.1.4 decontx

Runs the `decontx` workflow.

The pipelines generate the following files for each sample:

Table 2: Output Files

Output File	Description
out/data/.CELDA_DECONTX_[FILTER,CORRECTED].h5ad*	Contains the filtered matrix using one of the provided filters or the corrected (decontaminated) matrix by DecontX.
out/data/celda/.CELDA_DECONTX_R1.rds*	A file containing the SingleCellExperiment object processed by DecontX.
out/data/celda/.CELDA_DECONTX_Cell_Embeddings.tsv*	A cell-based TSV file containing data generated by DecontX and additional outlier masks: • decontX_contamination • decontX_clusters • celda_decontx__{doublemad,scater_isOutlier_3MAD,custom_gt_0.5}_predicted_outliers
out/data/celda/.CELDA_DECONTX_Outlier_Table.tsv*	A table containing the outlier thresholds for generating the outlier masks.
out/data/celda/.CELDA_DECONTX_Contamination_Density_Plot.pdf*	A plot showing the density of the decontamination scores from DecontX and the 3MAD area highlighted for the given outlier threshold.
out/data/celda/.CELDA_DECONTX_UMAP_Contamination.pdf*	A plot showing the DecontX contamination score on top of a UMAP generated from the decontaminated matrix.
out/data/celda/.CELDA_DECONTX_UMAP_Clusters.pdf*	A plot showing a UMAP generated by DecontX and from the decontaminated matrix.

5.1.5 single_sample_decontx

Runs the `single_sample` workflow above together with the DecontX workflow.

The DecontX workflow is running from the input data. The final processed file from the `single_sample` pipeline is annotated with the cell-based data generated by DecontX.

See `single_sample` and `decontx` to know more about the files generated by this pipeline.

5.1.6 single_sample_decontx_scrublet

Runs the `single_sample` workflow above together with the DecontX workflow.

The `single_sample` workflow is running from the input data. The `decontx` workflow is running from the input data. The `scrublet` workflow is running from the output of the DecontX workflow. The final processed file from the `single_sample` pipeline is annotated with the cell-based data generated by DecontX and Scrublet.

See `single_sample`, `decontx` and `scrublet` to know more about the files generated by this pipeline.

5.1.7 scenic

Runs the `scenic` workflow alone, generating a loom file with only the SCENIC results. Currently, the required input is a loom file (set by *params.tools.scenic.filteredLoom*).

5.1.8 scenic_multiruns

Runs the `scenic` workflow multiple times (set by `params.tools.scenic.numRuns`), generating a loom file with the aggregated results from the multiple SCENIC runs.

Note that this is not a complete entry-point itself, but a configuration option for the *scenic* module. Simply adding *-profile scenic_multiruns* during the config step will activate this analysis option for any of the standard entrypoints.

5.1.9 cellranger

Runs the `cellranger` workflow (`makefastq`, then `count`). Input parameters are specified within the config file:

- `params.tools.cellranger.mkfastq.csv`: path to the CSV samplesheet
- `params.tools.cellranger.mkfastq.runFolder`: path of Illumina BCL run folder
- `params.tools.cellranger.count.transcriptome`: path to the Cell Ranger compatible transcriptome reference

5.1.10 cellranger_count_metadata

Given the data stored as:

```
MKFASTQ_ID_SEQ_RUN1
|-- MAKE_FASTQS_CS
 -- outs
    |-- fastq_path
        |-- HFLC5BBXX
            |-- test_sample1
            |   |-- sample1_S1_L001_I1_001.fastq.gz
            |   |-- sample1_S1_L001_R1_001.fastq.gz
            |   |-- sample1_S1_L001_R2_001.fastq.gz
            |   |-- sample1_S1_L002_I1_001.fastq.gz
            |   |-- sample1_S1_L002_R1_001.fastq.gz
            |   |-- sample1_S1_L002_R2_001.fastq.gz
            |   |-- sample1_S1_L003_I1_001.fastq.gz
            |   |-- sample1_S1_L003_R1_001.fastq.gz
            |   |-- sample1_S1_L003_R2_001.fastq.gz
            |-- test_sample2
            |   |-- sample2_S2_L001_I1_001.fastq.gz
            |   |-- sample2_S2_L001_R1_001.fastq.gz
            |   |-- ...
        |-- Reports
        |-- Stats
        |-- Undetermined_S0_L001_I1_001.fastq.gz
        ...
        -- Undetermined_S0_L003_R2_001.fastq.gz
MKFASTQ_ID_SEQ_RUN2
|-- MAKE_FASTQS_CS
 -- outs
    |-- fastq_path
        |-- HFLY8GGLL
            |-- test_sample1
```

(continues on next page)

(continued from previous page)

```
            |   |-- ...
            |-- test_sample2
            |   |-- ...
        |-- ...
```

and a metadata table:

Table 3: Minimally Required Metadata Table

sample_name	fastqs_parent_dir_path	fastqs_dir_name	fastqs_sample_prefix	expect_cells
Sample1_Bio_Rep1	MKFASTQ_ID_SEQ_RUN1/outs/fastq_path/HFLxxxxxx	test_sample1	sample1	5000
Sample1_Bio_Rep1	MKFASTQ_ID_SEQ_RUN2/outs/fastq_path/HFLxxxxxx	test_sample1	sample1	5000
Sample1_Bio_Rep2	MKFASTQ_ID_SEQ_RUN1/outs/fastq_path/HFLxxxxxx	test_sample2	sample2	5000
Sample1_Bio_Rep2	MKFASTQ_ID_SEQ_RUN2/outs/fastq_path/HFLxxxxxx	test_sample2	sample2	5000

Optional columns:

- `short_uuid`: `sample_name` will be prefix by this value. This should be the same between sequencing runs of the same biological replicate
- `expect_cells`: This number will be used as argument for the `--expect-cells` parameter in `cellranger count`.
- `chemistry`: This chemistry will be used as argument for the `--chemistry` parameter in `cellranger count`.

and a config:

```
nextflow config \
   ~/vib-singlecell-nf/vsn-pipelines \
   -profile cellranger_count_metadata \
   > nextflow.config
```

and a workflow run command:

```
nextflow run \
    ~/vib-singlecell-nf/vsn-pipelines \
    -entry cellranger_count_metadata
```

The workflow will run Cell Ranger *count* on 2 samples, each using the 2 sequencing runs.

NOTES:

- If `fastqs_dir_name` does not exist, set it to `none`

5.1.11 demuxlet/freemuxlet

Runs the `demuxlet` or `freemuxlet` workflows (`dsc-pileup` [with prefiltering], then `freemuxlet` or `demuxlet`) Input parameters are specified within the config file:

- `params.tools.popscle.vcf`: path to the VCF file for demultiplexing
- `params.tools.popscle.freemuxlet.nSamples`: Number of clusters to extract (should match the number of samples pooled)
- `params.tools.popscle.demuxlet.field`: Field in the VCF with genotype information

5.1.12 nemesh

Runs the `nemesh` pipeline (Drop-seq) on a single sample or multiple samples separately.

Source

5.2 Sample Aggregation Pipelines

Pipelines to aggregate multiple datasets together.

5.2.1 bbknn

Runs the `bbknn` workflow (sample-specific filtering, merging of individual samples, normalization, log-transformation, HVG selection, PCA analysis, then the batch-effect correction steps: BBKNN, clustering, dimensionality reduction (UMAP only)). The output is a loom file with the results embedded.

Source: https://github.com/Teichlab/bbknn/blob/master/examples/pancreas.ipynb

Table 4: Output Files (not exhaustive list)

Output File	Description
out/data/.BBKNN.h5ad*	*Scanpy*-ready h5ad file containing all results. The *raw.X* slot contains the log-normalized data (if normalization & transformation steps applied) while the *X* slot contains the log-normalized scaled data.
out/data/.BBKNN.loom*	*SCope*-ready loom file containing all results.

5.2.2 bbknn_scenic

Runs the `bbknn` workflow above, then runs the `scenic` workflow on the output, generating a comprehensive loom file with the combined results. This could be very resource intensive, depending on the dataset.

Table 5: Output Files (not exhaustive list)

Output File	Description
out/data/.BBKNN.h5ad*	*Scanpy*-ready h5ad file containing all results from a *bbknn* workflow run. The *raw.X* slot contains the log-normalized data (if normalization & transformation steps applied) while the *X* slot contains the log-normalized scaled data.
out/data/.BBKNN_SCENIC.loom*	*SCope*-ready loom file containing all results from a *bbknn* workflow and a *scenic* workflow run (e.g.: regulon AUC matrix, regulons, . . .).

5.2.3 harmony

Runs the `harmony` workflow (sample-specific filtering, merging of individual samples, normalization, log-transformation, HVG selection, PCA analysis, batch-effect correction (Harmony), clustering, dimensionality reduction (t-SNE and UMAP)). The output is a loom file with the results embedded.

Table 6: Output Files (not exhaustive list)

Output File	Description
out/data/.HARMONY.h5ad*	*Scanpy*-ready h5ad file containing all results. The *raw.X* slot contains the log-normalized data (if normalization & transformation steps applied) while the *X* slot contains the log-normalized scaled data.
out/data/.HARMONY.loom*	*SCope*-ready loom file containing all results.

5.2.4 harmony_scenic

Runs the `harmony` workflow above, then runs the `scenic` workflow on the output, generating a comprehensive loom file with the combined results. This could be very resource intensive, depending on the dataset.

Table 7: Output Files (not exhaustive list)

Output File	Description
out/data/.HARMONY.h5ad*	*Scanpy*-ready h5ad file containing all results from a *harmony* workflow run. The *raw.X* slot contains the log-normalized data (if normalization & transformation steps applied) while the *X* slot contains the log-normalized scaled data.
out/data/.HARMONY_SCENIC.loom*	*SCope*-ready loom file containing all results from a *harmony* workflow and a *scenic* workflow run (e.g.: regulon AUC matrix, regulons, . . .).

5.2.5 mnncorrect

Runs the `mnncorrect` workflow (sample-specific filtering, merging of individual samples, normalization, log-transformation, HVG selection, PCA analysis, batch-effect correction (mnnCorrect), clustering, dimensionality reduction (t-SNE and UMAP)). The output is a loom file with the results embedded.

Table 8: Output Files (not exhaustive list)

Output File	Description
out/data/.MNNCORRECT.h5ad*	*Scanpy*-ready h5ad file containing all results. The *raw.X* slot contains the log-normalized data (if normalization & transformation steps applied) while the *X* slot contains the log-normalized scaled data.
out/data/.MNNCORRECT.loom*	*SCope*-ready loom file containing all results.

5.3 Utility Pipelines

Contrary to the aformentioned pipelines, these are not end-to-end. They are used to perform small incremental processing steps.

5.3.1 cell_annotate

Runs the `cell_annotate` workflow which will perform a cell-based annotation of the data using a set of provided .tsv metadata files. We show a use case here below with 10x Genomics data were it will annotate different samples using the `obo` method. For more information about this cell-based annotation feature please visit Cell-based metadata annotation section.

First, generate the config :

```
nextflow config \
   ~/vib-singlecell-nf/vsn-pipelines \
   -profile tenx,utils_cell_annotate,singularity
```

Make sure the following parts of the generated config are properly set:

```
[...]
data {
  tenx {
     cellranger_mex = '~/out/counts/*/outs/'
  }
}
tools {
    scanpy {
        container = 'vibsinglecellnf/scanpy:1.8.1'
    }
    cell_annotate {
        off = 'h5ad'
        method = 'obo'
        indexColumnName = 'BARCODE'
        cellMetaDataFilePath = "~/out/data/*.best"
        sampleSuffixWithExtension = '_demuxlet.best'
        annotationColumnNames = ['DROPLET.TYPE', 'NUM.SNPS', 'NUM.READS', 'SNG.BEST.
↪GUESS']
    }
    [...]
}
[...]
```

Now we can run it with the following command:

```
nextflow -C nextflow.config \
   run ~/vib-singlecell-nf/vsn-pipelines \
   -entry cell_annotate \
   > nextflow.config
```

5.3.2 cell_annotate_filter

Runs the `cell_annotate_filter` workflow which will perform a cell-based annotation of the data using a set of provided .tsv metadata files following by a cell-based filtering. We show a use case here below with 10x Genomics data

were it will annotate different samples using the `obo` method. For more information about this cell-based annotation feature please visit Cell-based metadata annotation section and Cell-based metadata filtering section.

First, generate the config :

```
nextflow config \
   ~/vib-singlecell-nf/vsn-pipelines \
   -profile tenx,utils_cell_annotate,utils_cell_filter,singularity \
   > nextflow.config
```

Make sure the following parts of the generated config are properly set:

```
[...]
data {
  tenx {
     cellranger_mex = '~/out/counts/*/outs/'
  }
}
tools {
    scanpy {
        container = 'vibsinglecellnf/scanpy:1.8.1'
    }
    cell_annotate {
        off = 'h5ad'
        method = 'obo'
        indexColumnName = 'BARCODE'
        cellMetaDataFilePath = "~/out/data/*.best"
        sampleSuffixWithExtension = '_demuxlet.best'
        annotationColumnNames = ['DROPLET.TYPE', 'NUM.SNPS', 'NUM.READS', 'SNG.BEST.
↪GUESS']
    }
    cell_filter {
        off = 'h5ad'
        method = 'internal'
        filters = [
            [
                id:'NO_DOUBLETS',
                sampleColumnName:'sample_id',
                filterColumnName:'DROPLET.TYPE',
                valuesToKeepFromFilterColumn: ['SNG']
            ]
        ]
    }
    [...]
}
[...]
```

Now we can run it with the following command:

```
nextflow -C nextflow.config \
   run ~/vib-singlecell-nf/vsn-pipelines \
   -entry cell_filter
```

5.3.3 sra

Runs the `sra` workflow which will download all (or user-defined selected) FASTQ files from a particular SRA project and format those with properly and human readable names.

First, generate the config :

```
nextflow config \
  ~/vib-singlecell-nf/vsn-pipelines \
    -profile sra,singularity \
    > nextflow.config
```

NOTES:

- The download of SRA files is by default limited to 20 Gb. If this limit needs to be increased please set `params.tools.sratoolkit.maxSize` accordingly. This limit can be 'removed' by setting the parameter to an arbitrarily high number (e.g.: 9999999999999).
- If you're a VSC user, you might want to add the `vsc` profile.
- The final output (FASTQ files) will available in `out/data/sra`
- If you're downloading 10x Genomics scATAC-seq data, make sure to set `params.tools.sratoolkit.includeTechnicalReads = true` and properly set `params.utils.sra_normalize_fastqs.fastq_read_suffixes`. In the case of downloading the scATAC-seq samples of SRP254409, `fastq_read_suffixes` would be set to `["R1", "R2", "I1", "I2"]`.

Now we can run it with the following command:

```
nextflow -C nextflow.config \
   run ~/vib-singlecell-nf/vsn-pipelines \
    -entry sra

$ nextflow -C nextflow.config run ~/vib-singlecell-nf/vsn-pipelines -entry sra
N E X T F L O W  ~  version 21.04.3
Launching `~/vib-singlecell-nf/vsn-pipelines/main.nf` [sleepy_goldstine] - revision:␣
↪ba1dedbf51
executor >  local (23)
[12/25b9d4] process > sra:DOWNLOAD_FROM_SRA:SRA_TO_METADATA (1)                    ␣
↪                     [100%] 1 of 1 _
[e2/d5a429] process > sra:DOWNLOAD_FROM_SRA:SRATOOLKIT__DOWNLOAD_FASTQS:DOWNLOAD_
↪FASTQS_FROM_SRA_ACC_ID (4) [ 33%] 3 of 9
[30/cba7a0] process > sra:DOWNLOAD_FROM_SRA:SRATOOLKIT__DOWNLOAD_FASTQS:FIX_AND_
↪COMPRESS_SRA_FASTQ (3)      [100%] 3 of 3
[76/97ce6e] process > sra:DOWNLOAD_FROM_SRA:NORMALIZE_SRA_FASTQS (3)               ␣
↪                     [100%] 3 of 3
[8c/3125c4] process > sra:PUBLISH:SC__PUBLISH (11)                                 ␣
↪                     [100%] 12 of 12
...
```

CHAPTER 6

Advanced Features

6.1 Two-pass strategy

Typically, cell- and gene-level filtering is one of the first steps performed in the analysis pipelines. This usually results in the pipeline being run in two passes. In the **first pass**, the default filters are applied (which are probably not valid for new datasets), and a separate QC report is generated for each sample. These QC reports can be inspected and the filters can be adjusted in the config file either for all samples (by editing the `params.tools.scanpy.filter` settings directly, or for individual samples by using the strategy described in multi-sample parameters. Then, the **second pass** restarts the pipeline with the correct filtering parameters applied (use `nextflow run ... -resume` to skip already completed steps).

6.1.1 Other notes

In order to run a specific pipeline (e.g. `single_sample`), the pipeline name must be specified as a **profile** when running `nextflow config ...` (so that the default parameters are included), and as the **entry** workflow when running the pipeline with `nextflow run`.

One exception to this is that the `-entry` pipeline can be one that is a subset of the one present in the config file. For example, in a pipeline with long running step that occurs after filtering (e.g. `single_sample_scenic`), it can be useful to generate the full config file (`nextflow config vib-singlecell-nf/vsn-pipelines -profile single_sample_scenic`), then run a first pass for filtering using `nextflow run vib-singlecell-nf/vsn-pipelines -entry single_sample`, and a second pass using the full pipeline `-entry single_sample_scenic`).

6.2 Avoid re-running SCENIC and use pre-existing results

Often one would like to test different batch effect correction methods with SCENIC. Naively, one would run the following commands:

```
nextflow config ~/vibsinglecellnf -profile tenx,bbknn,dm6,scenic,scenic_use_cistarget_
↪motifs,singularity > bbknn.config
nextflow -C bbknn.config run vib-singlecell-nf/vsn-pipelines -entry bbknn_scenic
```

and,

```
nextflow config ~/vibsinglecellnf -profile tenx,bbknn,dm6,scenic,scenic_use_cistarget_
↪motifs,singularity > bbknn.config
nextflow -C harmony.config run vib-singlecell-nf/vsn-pipelines -entry harmony_scenic
```

The annoying bit here is that we run SCENIC twice. This is what we would like to avoid since the SCENIC results will be the same. To avoid this one can run the following code for generating the *harmony_scenic.config*,

```
nextflow config ~/vibsinglecellnf -profile tenx,harmony,scenic_append_only,
↪singularity > harmony.config
```

This will add a different scenic entry in the config:

```
params {
    tools {
        scenic {
            container = 'vibsinglecellnf/scenic:0.11.2'
            report_ipynb = '/src/scenic/bin/reports/scenic_report.ipynb'
            existingScenicLoom = ''
            sampleSuffixWithExtension = '' // Suffix after the sample name in the␣
↪file path
            scenicoutdir = "${params.global.outdir}/scenic/"
            scenicScopeOutputLoom = 'SCENIC_SCope_output.loom'
        }
    }
}
```

Make sure that the following entries are correctly set before running the pipeline,

- `existingScenicLoom = ''`
- `sampleSuffixWithExtension = '' // Suffix after the sample name in the file path`

Finally run the pipeline,

```
nextflow -C harmony.config run vib-singlecell-nf/vsn-pipelines -entry harmony_scenic
```

6.3 Set the seed

Some steps in the pipelines are non-deterministic. In order to have reproducible results, a seed is set by default to:

```
workflow.manifest.version.replaceAll("\\.","").toInteger()
```

The seed is a number derived from the version of the pipeline used at the time of the analysis run. To override the seed (integer) you have edit the `nextflow.config` file with:

```
params {
    global {
        seed = [your-custom-seed]
```

(continues on next page)

(continued from previous page)

```
    }
}
```

This filter will only be applied on the final loom file of the VSN-Pipelines. All the intermediate files prior to the loom file will still contain all of them the markers.

6.4 Change log fold change (logFC) and false discovery rate (FDR) thresholds for the marker genes stored in the final SCope loom

By default, the logFC and FDR thresholds are set to 0 and 0.05 respectively. If you want to change those thresholds applied on the markers genes, edit the `nextflow.config` with the following entries,

```
params {
    tools {
        scope {
            markers {
                log_fc_threshold = 0.5
                fdr_fc_threshold = 0.01
            }
        }
    }
}
```

This filter will only be applied on the final loom file of the VSN-Pipelines. All the intermediate files prior to the loom file will still contain all of them the markers.

6.5 Automated selection of the optimal number of principal components

When generating the config using `nextflow config` (see above), add the `pcacv` profile.

Remarks:

- Make sure `nComps` config parameter (under `dim_reduction.pca`) is not set.
- If `nPcs` is not set for t-SNE or UMAP config entries, then all the PCs from the PCA will be used in the computation.

Currently, only the Scanpy related pipelines have this feature implemented.

6.6 Cell-based metadata annotation

There are 2 ways of using this feature: either when running an end-to-end pipeline (e.g.: `single_sample`, `harmony`, `bbknn`, ...) or on its own as a independent workflow.

The profile `utils_cell_annotate` should be added along with the other profiles when generating the main config using the `nextflow config` command.

For more detailed information about those parameters, please check the **'cell_annotate parameter details <Parameters of cell_annotate_>'_** section below.

Please check the cell_annotate workflow.

The `utils_cell_annotate` profile is adding the following part to the config:

```
params {
    tools {
        cell_annotate {
            off = 'h5ad'
            method = ''
            cellMetaDataFilePath = ''
            sampleSuffixWithExtension = ''
            indexColumnName = ''
            sampleColumnName = ''
            annotationColumnNames = ['']
        }
    }
}
```

Two methods (`params.utils.cell_annotate.method`) are available:

- `aio`
- `obo`

If you have a single file containing the metadata information of all your samples, use `aio` method otherwise use `obo`.

For both methods, here are the mandatory parameters to set:

- `off` should be set to `h5ad`
- `method` choose either `obo` or `aio`
- `annotationColumnNames` is an array of columns names from `cellMetaDataFilePath` containing different annotation metadata to add.

If `aio` used, the following additional parameters are required:

- `cellMetaDataFilePath` is a file path pointing to a single .tsv file (with header) with at least 2 columns: a column containing all the cell IDs and an annotation column.
- `indexColumnName` is the column name from `cellMetaDataFilePath` containing the cell IDs information. This column **can** have unique values; if it's not the case, it's important that the combination of the values from the `indexColumnName` and the `sampleColumnName` are unique.
- `sampleColumnName` is the column name from `cellMetaDataFilePath` containing the sample ID/name information. Make sure that the values from this column match the samples IDs inferred from the data files. To know how those are inferred, please read the Input Data Formats section.

If `obo` is used, the following parameters are required:

- `cellMetaDataFilePath`
  - In multi-sample mode, is a file path containing a glob pattern. The target file paths should each pointing to a .tsv file (with header) with at least 2 columns: a column containing all the cell IDs and an annotation column.
  - In single-sample mode, is a file path pointing to a single .tsv file (with header) with at least 2 columns: a column containing all the cell IDs and an annotation column.
  - **Note**: the file name(s) of `cellMetaDataFilePath` is/are required to contain the sample ID(s).
- `sampleSuffixWithExtension` is the suffix used to extract the sample ID from the file name(s) of `cellMetaDataFilePath`. The suffix should be the part after the sample name in the file path.

- `indexColumnName` is the column name from `cellMetaDataFilePath` containing the cell IDs information. This column **must** have unique values.

6.7 Sample-based metadata annotation

The profile `utils_sample_annotate` should be added when generating the main config using `nextflow config`. This will add the following entry in the config:

```
params {
    tools {
        sample_annotate {
            iff = '10x_cellranger_mex'
            off = 'h5ad'
            type = 'sample'
            metadataFilePath = 'data/10x/1k_pbmc/metadata.tsv'
        }
    }
}
```

Then, the following parameters should be updated to use the module feature:

- `metadataFilePath` is a .tsv file (with header) with at least 2 columns where the first column need to match the sample IDs. Any other columns will be added as annotation in the final loom i.e.: all the cells related to their sample will get annotated with their given annotations.

Table 1: Sample-based Metadata Table

id	chemistry	...
1k_pbmc_v2_chemistry	v2	...
1k_pbmc_v3_chemistry	v3	...

Sample-annotating the samples using this system will allow any user to query all the annotation using the SCope portal. This is especially relevant when samples needs to be compared across specific annotations (check compare tab with SCope).

6.8 Cell-based metadata filtering

There are 2 ways of using this feature: either when running an end-to-end pipeline (e.g.: `single_sample`, `harmony`, `bbknn`, ...) or on its own as a independent workflow.

The `utils_cell_filter` profile is required when generating the config file. This profile will add the following part:

```
params {
    tools {
        cell_filter {
            off = 'h5ad'
            method = ''
            filters = [
                [
                    id: '',
                    sampleColumnName: '',
                    filterColumnName: '',
```

(continues on next page)

(continued from previous page)

```
                    valuesToKeepFromFilterColumn: ['']
                ]
            ]
        }
    }
}
```

For more detailed information about the parameters to set in `params.utils.cell_filter`, please check the cell_filter parameter details section below.

Please check the cell_filter workflow or cell_annotate_filter workflow to perform cell-based annotation and cell-based filtering sequentially.

Two methods (`params.utils.cell_filter.method`) are available:

- `internal`
- `external`

If you have a single file containing the metadata information of all your samples, use `external` method otherwise use `internal`.

For both methods, here are the mandatory parameters to set:

- `off` should be set to `h5ad`
- `method` choose either `internal` or `external`
- `filters` is a List of Maps where each Map is required to have the following parameters:
  - `id` is a short identifier for the filter
  - `valuesToKeepFromFilterColumn` is array of values from the `filterColumnName` that should be kept (other values will be filtered out).

If `internal` used, the following additional parameters are required:

- `filters` is a List of Maps where each Map is required to have the following parameters:
  - `sampleColumnName` is the column name containing the sample ID/name information. It should exist in the `obs` column attribute of the h5ad.
  - `filterColumnName` is the column name that will be used to filter out cells. It should exist in the `obs` column attribute of the h5ad.

If `external` used, the following additional parameters are required:

- `filters` is a List of Maps where each Map is required to have the following parameters:
  - `cellMetaDataFilePath` is a file path pointing to a single .tsv file (with header) with at least 3 columns: a column containing all the cell IDs, another containing the sample ID/name information, and a column to use for the filtering.
  - `indexColumnName` is the column name from `cellMetaDataFilePath` containing the cell IDs information. This column **must** have unique values.
  - *optional* `sampleColumnName` is the column name from `cellMetaDataFilePath` containing the sample ID/name information. Make sure that the values from this column match the samples IDs inferred from the data files. To know how those are inferred, please read the Input Data Formats section.
  - *optional* `filterColumnName` is the column name from `cellMetaDataFilePath` which be used to filter out cells.

6.9 Multi-sample parameters

It's possible to define custom parameters for the different samples. It's as easy as defining a hashmap in groovy or a dictionary-like structure in Python. You'll just have to repeat the following structure for the parameters which you want to enable the multi-sample feature for

```
params {
    tools {
        scanpy {
        container = 'vibsinglecellnf/scanpy:1.8.1'
        filter {
            report_ipynb = '/src/scanpy/bin/reports/sc_filter_qc_report.ipynb'
            // Here we enable the multi-sample feature for the cellFilterMinNgenes␣
↪parameter
            cellFilterMinNGenes = [
                '1k_pbmc_v2_chemistry': 600,
                '1k_pbmc_v3_chemistry': 800
            ]
            // cellFilterMaxNGenes will be set to 4000 for all the samples
            cellFilterMaxNGenes = 4000
            // Here we again enable the multi-sample feature for the␣
↪cellFilterMaxPercentMito parameter
            cellFilterMaxPercentMito = [
                '1k_pbmc_v2_chemistry': 0.15,
                '1k_pbmc_v3_chemistry': 0.05
            ]
            // geneFilterMinNCells will be set to 3 for all the samples
            geneFilterMinNCells = 3
            iff = '10x_mtx'
            off = 'h5ad'
            outdir = 'out'
        }
    }
}
```

If you want to apply custom parameters for some specific samples and have a "general" parameter for the rest of the samples, you should use the 'default' key as follows:

```
params {
    tools {
        scanpy {
        container = 'vibsinglecellnf/scanpy:1.8.1'
        filter {
            report_ipynb = '/src/scanpy/bin/reports/sc_filter_qc_report.ipynb'
            // Here we enable the multi-sample feature for the cellFilterMinNgenes␣
↪parameter
            cellFilterMinNGenes = [
                '1k_pbmc_v2_chemistry': 600,
                'default': 800
            ]
            [...]
        }
    }
}
```

Using this config, the parameter `params.tools.scanpy.cellFilterMinNGenes` will be applied with a threshold value of `600` to `1k_pbmc_v2_chemistry`. The rest of the samples will use the value `800` to filter

the cells having less than that number of genes. This strategy can be applied to any other parameter of the config.

6.10 Parameter exploration

Since `v0.9.0`, it is possible to explore several combinations of parameters. The latest version of the VSN-Pipelines allows to explore the following parameters:

- `params.tools.scanpy.clustering`
  - `method`

```
methods = ['louvain','leiden']
```

  - `resolution`

```
resolutions = [0.4, 0.8]
```

In case the parameter exploration mode is used within the `params.tools.scanpy.clustering` parameter, it will generated a range of different clusterings. For non-expert, it's often difficult to know which clustering to pick. It's however possible to use the `DIRECTS` module in order to select a default clustering. In order, to use this automated clustering selection method, add the `directs` profile when generating the main config using `nextflow config`. The config will get populated with:

```
directs {
    container = 'vibsinglecellnf/directs:0.1.0'
    labels {
        processExecutor = 'local'
    }
    select_default_clustering {
        fromMinClusterSize = 5
        toMinClusterSize = 100
        byMinClusterSize = 5
    }
}
```

Currently, only the Scanpy related pipelines have this feature implemented.

6.11 Regress out variables

By default, don't regress any variable out. To enable this features, the `scanpy_regress_out` profile should be added when generating the main config using `nextflow config`. This will add the following entry in the config:

```
params {
    tools {
        scanpy {
            regress_out {
                variablesToRegressOut = []
                off = 'h5ad'
            }
        }
    }
}
```

Add any variable in `variablesToRegressOut` to regress out: e.g.: 'n_counts', 'percent_mito'.

6.12 Highly Variable Genes Selection

This step is a wrapper around the *Scanpy* `scanpy.pp.highly_variable_genes` function and regarding the parameters used it is following the documentation available at scanpy-pp-highly-variable-genes. By default, it will use the `seurat` flavor to select variable genes and will also keep the same default values for the 4 different thresholds (as the documentation): `min_mean`, `max_mean`, `min_disp`, `max_disp`.

```
params {
    tools {
        scanpy {
            feature_selection {
                report_ipynb = "${params.misc.test.enabled ? '../../..' : ''}/src/
↪scanpy/bin/reports/sc_select_variable_genes_report.ipynb"
                flavor = 'seurat'
                minMean = 0.0125
                maxMean = 3
                minDisp = 0.5
                off = 'h5ad'
            }
        }
    }
}
```

Other flavors are available as `cell_ranger` and `seurat_v3`. In order to use the `seurat_v3` flavor, one parameter is required to be specified: `nTopGenes` in the config file as follows:

```
params {
    tools {
        scanpy {
            feature_selection {
                report_ipynb = "${params.misc.test.enabled ? '../../..' : ''}/src/
↪scanpy/bin/reports/sc_select_variable_genes_report.ipynb"
                flavor = 'seurat_v3'
                nTopGenes = 2000
                off = 'h5ad'
            }
        }
    }
}
```

6.13 Skip steps

By default, the pipelines are run from raw data (unfiltered data, not normalized).

If you have already performed an independent steps with another it's possible to skip some steps from the pipelines. Currently, here are the steps that can be skipped: - `Scanpy` filtering - `Scanpy` normalization

In order to skip the Scanpy filtering step, we need to add 3 new profiles when generating the config:

- `min`
- `scanpy_data_transformation`
- `scanpy_normalization`

The following command, will create a Nextflow config which the pipeline will understand and will not run the Scanpy filtering step:

```
nextflow config \
   ~/vib-singlecell-nf/vsn-pipelines \
   -profile min,[data-profile],scanpy_data_transformation,scanpy_normalization,[...],
↪singularity \
   > nextflow.config
```

- `[data-profile]`: Can be one of the different possible data profiles e.g.: `h5ad`
- `[...]`: Can be other profiles like `bbknn`, `harmony`, `pcacv`, ...

6.14 Quiet mode

By default, VSN will output some additional messages to the terminal, such as the global seed, and the names and paths of the samples detected by the input channel. These messages can be suppressed by using the `--quiet` flag when starting the nextflow process:

```
nextflow -C example.config run vib-singlecell-nf/vsn-pipelines -entry single_sample --
↪quiet
```

CHAPTER 7

Case Studies

See the full list of case studies and examples at VSN-Pipelines-examples.

CHAPTER 8

scATAC-seq Preprocessing

This pipeline takes fastq files from paired end single cell ATAC-seq, and applies preprocessing steps to align the reads to a reference genome, and produce a bam file and scATAC-seq fragments file.

This workflow is currently available in the `develop_atac` branch (use `nextflow pull vib-singlecell-nf/vsn-pipelines -r develop_atac` to sync this branch).

8.1 Pipeline Steps

The full steps are:

- Barcode correction:
  - For 'standard' and 'multiome' samples (e.g. 10x Genomics or similar) correction is performed against a whitelist by this method from aertslab/single_cell_toolkit.
  - For 'biorad' samples, barcode correction is performed by this script in our aertslab/single_cell_toolkit (previously, this was done with BAP).
- Fastq barcoding: Add the barcode sequence to the comment field of the fastq sequence identifier. Uses methods from aertslab/single_cell_toolkit.
- Read/adapter trimming (Trim_Galore or fastp).
- Mapping to a reference genome:
  - `bwa mem` is used with default parameters, with a choice of the original bwa mem, or bwa-mem2.
- Mark PCR and optical duplicates (MarkDuplicates (Picard) or MarkDuplicatesSpark (GATK)).
- Estimate library complexity with EstimateLibraryComplexity (Picard).
- A fragments file is created using Sinto.

8.2 Pipeline Details

8.2.1 Input Metadata

The input to this pipeline is a (tab-delimited) metadata table with the sample ID, sequencing technology, and locations of the fastq files. Note that the fastq file fields must be full paths; this is not shown here for clarity:

Table 1: Metadata Table

sample_name	technology	fastq_PE1_path	fastq_barcode_path	fastq_PE2_path
sample_1	standard	sample_1_R1.fastq.gz	sample_1_R2.fastq.gz	sample_1_R3.fastq.gz
sample_2	multiome	sample_2_R1.fastq.gz	sample_2_R2.fastq.gz	sample_2_R3.fastq.gz
sample_3	biorad	sample_3_R1.fastq.gz		sample_3_R3.fastq.gz
sample_4	revcomp_wl	sample_4_R1.fastq.gz	sample_2_R2.fastq.gz	sample_4_R3.fastq.gz
sample_5	hydrop_3x96	sample_5_R1.fastq.gz	sample_5_R2.fastq.gz	sample_5_R3.fastq.gz
sample_6	hydrop_2x384	sample_5_R1.fastq.gz	sample_5_R2.fastq.gz	sample_5_R3.fastq.gz

The columns represent:

- `sample_name` Sample name for labeling the sample in the pipeline and output files. This can be any arbitrary string.
- `technology`: This controls the barcode correction and processing methods to use for the fastq files. Currently only the `biorad` option involves different processing steps. Otherwise, the value in this field (e.g. `standard`, `multiome`) controls which barcode whitelist is used for correction. See below for additional details.
- `fastq_PE1_path`: The full path to the fastq file for the first read in a pair.
- `fastq_barcode_path`: The full path to the fastq file containing the barcodes. This column can be blank/empty depending on the technology setting (e.g. `biorad`).
- `fastq_PE2_path`: The full path to the fastq file for the second read in a pair.

8.2.2 Fastq input

Fastq input for each sample can be given as a single set of files (R1, R2, R3), or it can be multiple files, in the case of samples which have been split across multiple sequencing lanes. A combination of these cases can be processed together in one metadata file.

Within the pipeline, all of the reads from each set of files will be considered and labeled as one read group. Read group names are taken from the first read in the fastq:

```
@A01044:19:HLYKFDRXX:1:2101:4291:1000 1:N:0:ACTCAGAC
```

will produce a RG in the bam:

```
@RG     ID:A01044:19:HLYKFDRXX:1        SM:sample_1        LB:A01044:19:HLYKFDRXX:1__
↪sample_1 PL:ILLUMINA
```

Single fastq input

In this situation, there is only one set of reads per sample, the metadata file will look very similar to one of the rows from above. There will be one read group in the final bam file.

Split fastq input

In this case, multiple fastq files (in rows) for each sample can be given. In this example, there are two sets of fastqs for `sample_1` that were run on two separate lanes. Note that the sample ID is the same for both rows:

Table 2: Metadata Table with split fastqs

sample_name	technology	fastq_PE1_path	fastq_barcode_path	fastq_PE2_path
sample_1	standard	sample_1_S1_L001_R1_001.fastq.gz	sample_1_S1_L001_R2_001.fastq.gz	sample_1_S1_L001_R3_001.fastq.gz
sample_1	standard	sample_1_S1_L002_R1_001.fastq.gz	sample_1_S1_L002_R2_001.fastq.gz	sample_1_S1_L002_R3_001.fastq.gz

In this situation, each set of fastqs will be processed separately for the barcode correction, barcode addition to the fastq comment field, adaptor trimming, and mapping steps. Following mapping, each mapped bam is merged and duplicates are marked using the full data. Downstream steps are done with the merged data.

8.2.3 Generating the metadata file

Note that there is an easy way to create the metadata from the file paths for each sample by using the following bash command (expand to view). Special thanks here to Gert Hulselmans for expanding the capabilities of this function.

```
create_atac_metadata () {
    local sample="${1}";
    local technology="${2}";
    local fastq_prefix="${3}";
    local read_labels="${4}";
    if [ "${sample}" == "header" ]; then
        printf 'sample_name\ttechnology\tfastq_PE1_path\tfastq_barcode_path\tfastq_
↪PE2_path\n';
        return 0;
    fi
    if [ ${#@} -ne 4 ] ; then
        printf 'Usage: create_atac_metadata sample technology fastq_prefix read_
↪labels\n\n';
        printf 'Arguments:\n';
        printf '    sample:       sample name\n';
        printf '    technology:   "standard", "hydrop_3x96", "hydrop_2x384", or
↪"biorad"\n';
        printf '    fastq_prefix: path prefix to FASTQ files.\n';
        printf '    read_labels:  comma separated read labels for R1, R2 and R3 that␣
↪select: R1,R2,R3.\n';
        return 1;
    fi
    read_labels_arr=(${read_labels//,/ });
    # Get R1, R2 and R3 FASTQ filenames for
    R1=(${fastq_prefix}*${read_labels_arr[0]}*.{fastq,fq,fastq.gz,fq.gz})
    R2=(${fastq_prefix}*${read_labels_arr[1]}*.{fastq,fq,fastq.gz,fq.gz})
    R3=(${fastq_prefix}*${read_labels_arr[2]}*.{fastq,fq,fastq.gz,fq.gz})
    for i in "${!R1[@]}" ; do
        # Check if R1 FASTQ file exist (and is not just a glob like "${sample}*R1*.fq
↪").
        if [ -e "${R1[i]}" ] ; then
            printf "${sample}\t${technology}\t${R1[i]}\t${R2[i]}\t${R3[i]}\n";
        fi
    done
}
```

To run use the options:

1. Sample ID (if this parameter is "header", it will print the metadata header and stop)
2. Technology (e.g. "standard")
3. The "file prefix" full path to your fastq files, matching the common portions of the file names (without any glob * expansions)
4. The "read labels" to indicate how the files are named and match the remainder of the file names (e.g. "R1,R2,R3", "R1,UMI,R2", etc.)

```
create_atac_metadata header > auto_metadata.tsv
create_atac_metadata sample_1 standard /path/to/sample_1_subset_S R1,R2,R3 >> auto_
↪metadata.tsv
create_atac_metadata sample_2 standard /path/to/sample_2_subset_S R1,R2,R3 >> auto_
↪metadata.tsv
create_atac_metadata sample_5 hydrop_3x96 /path/to/sample_5_ R1,R2,R3 >> auto_
↪metadata.tsv
create_atac_metadata sample_6 hydrop_2x384 /path/to/sample_6_ R1,R2,R3 >> auto_
↪metadata.tsv
```

8.2.4 Technology types

The "technology" field in the metadata table controls two things:

1. **How technology-specific pipeline steps are applied.** Currently there are three specific settings (`biorad`, `hydrop_3x96`, and `hydrop_2x384`) that use alternate pipelines processes (to extract and correct the barcode sequence from the input fastqs). Using any other keyword is allowed, and samples will be run with the standard pipeline steps (barcode correction against a whitelist).
2. **Which whitelist is used for barcode correction.** The "technology" field must match a key in the `params.tools.singlecelltoolkit.barcode_correction.whitelist` parameter list in the config file for that sample to be associated with a particular barcode whitelist. The "technology" field and whitelist key name can be set to any arbitrary string (e.g. `standard`), with the exception of the technology-specific keywords above.

The main modes are:

standard

The `standard` setting is the main pipeline mode. It assumes a typical 10x Genomics style format with two read pair fastqs and a barcode fastq (note that in the example here, the barcode correction has already been performed, writing the `CB` tag into the comment of the barcode fastq):

```
$ zcat sample_1_R1.fastq.gz | head -n 4
@A00311:74:HMLK5DMXX:1:1101:2013:1000 1:N:0:ACTCAGAC
NTTGTCTCAGCACCCCCCGACATGGATTCAGGCTGTCTCTTATACACATC
+
#FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

$ zcat sample_1_R2.fastq.gz | head -n 4
@A00311:74:HMLK5DMXX:1:1101:2013:1000 2:N:0:ACTCAGAC CB:Z:CTGTTCGCAAAGCATA
CTGTTCGCAAAGCATA
+
F:FFFFFFFFFFFFFF
```

(continues on next page)

(continued from previous page)

```
$ zcat sample_1_R3.fastq.gz | head -n 4
@A00311:74:HMLK5DMXX:1:1101:2013:1000 3:N:0:ACTCAGAC
CCTGAATCCATGTCGGGGGGTGCTGAGACAAGCTGTCTCTTATACACAT
+
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

The barcoding step here uses a helper script from aertslab/single_cell_toolkit which transforms this input into two paired fastq files with the barcode information embedded in the fastq comments field:

```
$ zcat sample_1_dex_R1.fastq.gz | head -n 4
@A00311:74:HMLK5DMXX:1:1101:2013:1000 CR:Z:CTGTTCGCAAAGCATA     CY:Z:F:FFFFFFFFFFFFFF␣
↪  CB:Z:CTGTTCGCAAAGCATA
NTTGTCTCAGCACCCCCCGACATGGATTCAGGCTGTCTCTTATACACATC
+
#FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

$ zcat sample_1_dex_R2.fastq.gz | head -n 4
@A00311:74:HMLK5DMXX:1:1101:2013:1000 CR:Z:CTGTTCGCAAAGCATA     CY:Z:F:FFFFFFFFFFFFFF␣
↪  CB:Z:CTGTTCGCAAAGCATA
CCTGAATCCATGTCGGGGGGTGCTGAGACAAGCTGTCTCTTATACACAT
+
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

`multiome`/alternate

The `multiome` or alternately-named settings work with the same pipeline steps as `standard` with the exception of the whitelist used for barcode correction. The whitelists are supplied in the params file (`params.tools.singlecelltoolkit.barcode_correction.whitelist`). This can be used to supply alternate whitelists for certain samples, for example if you need to supply a reverse complemented whitelist for samples run in certain sequencing machines.

`hydrop_3x96`/`hydrop_2x384`

The HyDrop settings (either `hydrop_3x96` or `hydrop_2x384` depending on the library preparation used) processes data generated by the HyDrop ATAC protocol (see hydrop.aertslab.org and the associated preprint). This approach differs from the standard pipeline in only the initial step, which is to extract and process the HyDrop barcodes from the sequencing output. Here, this script is used to take the R2 read from the sequencer:

```
$ zcat sample_5_R2.fastq.gz | head -n 4
@VH00445:5:AAAL5KYM5:1:1101:63923:1019 2:N:0:ACACGTGGAC
CACTGGTGGTAGGGTACTCGGACAAGTGGAGCAGTAGCTGAAGTGTAGAAG
+
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

and transform it into:

```
$ zcat sample_5_hydrop_barcode_R2.fastq.gz
@VH00445:5:AAAL5KYM5:1:1101:63923:1019 2:N:0:ACACGTGGAC
CACTGGTGGTGACAAGTGGAAAGTGTAGAA
+
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

The two HyDrop modes (`hydrop_3x96`, `hydrop_2x384`) differ only in the way the initial barcode extraction script works. Following this, they are processed in the same way as the standard pipeline, including whitelist-based barcode correction (note that the two HyDrop modes require different barcode whitelists to be used here).

biorad

The `biorad` setting processes BioRad data using this script in our aertslab/single_cell_toolkit (previously, this was done with BAP). This takes input data:

```
$ zcat sample_2_R1.fastq.gz | head -n 4
@A00794:327:HTJ55DRXX:1:2101:1154:1016 1:N:0:TAAGGCGA
GATCACCATATGCATGACATTCACGAGTCACTGAGTAACGCCTCGTCGGCAGCGTCAGATGTGTATAAGAGACAGCTGCAATGGCTGGAGCACACCCCATA
+
FFFFFFFFFFFFFFFFFFFFFFFFFF:FFFFFFF:FFFFFFFFF:FFFFFFFFFFFFFFFF,FF,
↪FFFFFFFFF:FFFFFFFFFFFFFFFFFFFFFFFFFF:F:FFFF,FFFFFF

$ zcat sample_2_R2.fastq.gz | head -n 4
@A00794:327:HTJ55DRXX:1:2101:1154:1016 2:N:0:TAAGGCGA
GTGTTTGGCTGAGGAAAGTGTGTGAAGCCCCGATATGTGA
+
FFF,FFF:FFF:FF,FFFFF:F:FFFFFFFFFFF,,F:FF
```

and directly produces paired fastq files with the barcode added in the fastq comments field:

```
$ zcat sample_2_dex_R1.fastq.gz | head -n 4
@A00794:327:HTJ55DRXX:1:2101:1154:1016 CR:Z:GATCACCATTCACGTAACGCC
↪CY:Z:FFFFFFFFFFFFFF:FFFFFF      CB:Z:GATCACCATTCACGTAACGCC      br:Z:0,0,0_0,0,0,1
CTGCAATGGCTGGAGCACACCCCATACTCATTCTGGTCTCCTT
+
F:FFFFFFFFFFFFFFFFFFFFFFFFFF:F:FFFF,FFFFFF

$ zcat sample_2_dex_R2.fastq.gz | head -n 4
@A00794:327:HTJ55DRXX:1:2101:1154:1016 CR:Z:GATCACCATTCACGTAACGCC
↪CY:Z:FFFFFFFFFFFFFF:FFFFFF      CB:Z:GATCACCATTCACGTAACGCC      br:Z:0,0,0_0,0,0,1
GTGTTTGGCTGAGGAAAGTGTGTGAAGCCCCGATATGTGA
+
FFF,FFF:FFF:FF,FFFFF:F:FFFFFFFFFFF,,F:FF
```

8.3 Running the workflow

8.3.1 Technical considerations

1. Direct the Nextflow work directory to an alternate path (e.g. a scratch drive) using the `NXF_WORK` environmental variable:

```
nwork=/path/to/scratch/example_project
mkdir $nwork
export NXF_WORK=$nwork
```

Note that if you start a new shell, `NXF_WORK` must be set again, or the pipeline will not resume properly.

2. Temporary directory mapping. For large BAM files, the system default temp location may become full. A workaround is to include a volume mapping to the alternate `/tmp -B /alternate/path/to/tmp:/tmp` using the volume mount options in Docker or Singularity. For example in the container engine options:

- Singularity run options: `runOptions = '--cleanenv -H $PWD -B /data,/alternate/path/to/tmp:/tmp'`
- Docker run options: `runOptions = '-i -v /data:/data -v /alternate/path/to/tmp:/tmp'`

8.3.2 Configuration

To generate a config file, use the `atac_preprocess` profile along with `docker` or `singularity`. Note that the full path to `vib-singlecell-nf/vsn-pipelines/main_atac.nf` must be used:

```
nextflow config \
    vib-singlecell-nf/vsn-pipelines/main_atac.nf \
    -profile atac_preprocess,singularity \
    > atac_preprocess.config
```

Note: It is also possible to run the pycisTopic QC steps directly after this `atac_preprocess` pipeline, with a single command. Please see here for details on how to run with this configuration.

8.3.3 Parameters

The ATAC-specific parameters are described here. The important parameters to verify are:

- `params.data.atac_preprocess.metadata`: the path to the metadata file.
- `params.tools.bwamaptools.bwa_fasta`: the path to the bwa reference fasta file. This should be already indexed with `bwa index`, and the index files located in the same directory as the fasta file. Note that `bwa` and `bwa-mem2` use different indexes that are not interchangeable.
- `params.tools.singlecelltoolkit.barcode_correction.whitelist`: Whitelists for barcode correction are supplied here. The whitelists are matched to samples based on the parameter key here ('standard', 'multiome', 'hydrop_3x96', 'hydrop_2x384', etc.) and the technology field listed for each sample in the metadata file. Barcode whitelists can (optionally) be gzipped. There are currently no checks performed to ensure that the sample barcodes have any overlap to the whitelist (the barcode correction reports should be checked for this).

Choice of tools

Several steps have options for the choice of method to use. These options are controlled within `params.atac_preprocess_tools`.

- Adapter trimming (`adapter_trimming_method`): Can be either of `Trim_Galore` (default), or `fastp`.
- Duplicate marking (`mark_duplicates_method`): Can be either of `MarkDuplicates` (Picard tools, default) or `MarkDuplicatesSpark` (GATK). We currently recommend Picard MarkDuplicates because it has the capability to perform barcode-aware marking of PCR duplicates. MarkDuplicatesSpark has the advantage of parallelization, however it requires a large SSD to use for temporary files.

Additionally:

- Mapping: Use parameter `params.tools.bwamaptools.bwa_version` to select either `bwa` or `bwa-mem2`. These should give virtually identical results, however `bwa-mem2`, while faster, has used more memory in our tests. Note that the index (`bwa_index`) is not interchangeable between the versions.

Optional parameters

- Within `params.tools.sinto.fragments`:
  - One of (but not both) `barcodetag` or `barcode_regex` needs to be set to tell Sinto where to find the barcodes in the bam file. The default is to use `barcodetag` of `CB`.
  - `mapq`: Controls quality filtering settings for generating the fragments file. Discards reads with quality score lower than this number (default 30).

8.3.4 Execution

After configuring, the workflow can be run with:

```
nextflow -C atac_preprocess.config run \
    vib-singlecell-nf/vsn-pipelines/main_atac.nf \
    -entry atac_preprocess -resume
```

8.4 Output

An example output tree is shown here.

```
out/
├── data
│   ├── bam
│   │   ├── sample_1.bwa.out.possorted.bam
│   │   ├── sample_1.bwa.out.possorted.bam.bai
│   │   ├── sample_2.bwa.out.possorted.bam
│   │   └── sample_2.bwa.out.possorted.bam.bai
│   ├── fragments
│   │   ├── sample_1.sinto.fragments.tsv.gz
│   │   ├── sample_1.sinto.fragments.tsv.gz.tbi
│   │   ├── sample_2.sinto.fragments.tsv.gz
│   │   └── sample_2.sinto.fragments.tsv.gz.tbi
│   └── reports
│       ├── barcode
│       │   ├── sample_1____S7_R1_001.corrected.bc_stats.log
│       │   └── sample_2____S8_R1_001.corrected.bc_stats.log
│       ├── mapping_stats
│       │   ├── sample_1.mapping_stats.tsv
│       │   └── sample_2.mapping_stats.tsv
│       ├── mark_duplicates
│       │   ├── sample_1.library_complexity_metrics.txt
│       │   ├── sample_1.mark_duplicates_metrics.txt
│       │   ├── sample_2.library_complexity_metrics.txt
│       │   └── sample_2.mark_duplicates_metrics.txt
│       └── trim
│           ├── sample_1____S7_R1_001.fastp.trimming_report.html
│           └── sample_2____S8_R1_001.fastp.trimming_report.html
│
```

(continues on next page)

(continued from previous page)

```
└── nextflow_reports
    ├── execution_report.html
    ├── execution_timeline.html
    ├── execution_trace.txt
    └── pipeline_dag.dot
```

CHAPTER 9

scATAC-seq QC and Cell Calling

This workflow uses the Python implementation of cisTopic (pycisTopic) to perform quality control and cell calling. The inputs here are a fragments and bam file for each sample.

This workflow is currently available in the `develop_atac` branch (use `nextflow pull vib-singlecell-nf/vsn-pipelines -r develop_atac` to sync this branch).

9.1 Running the workflow

9.1.1 Technical considerations

1. Direct the Nextflow work directory to an alternate path (e.g. a scratch drive) using the `NXF_WORK` environmental variable:

```
nwork=/path/to/scratch/example_project
mkdir $nwork
export NXF_WORK=$nwork
```

Note that if you start a new shell, `NXF_WORK` must be set again, or the pipeline will not resume properly.

2. Important for pycisTopic Ray issues: the system default temp location may become full. A workaround is to include a volume mapping to the alternate `/tmp -B /alternate/path/to/tmp:/tmp` using the volume mount options in Docker or Singularity. For example in the container engine options:

- Singularity run options: `runOptions = '--cleanenv -H $PWD -B /data,/alternate/path/to/tmp:/tmp'`
- Docker run options: `runOptions = '-i -v /data:/data -v /alternate/path/to/tmp:/tmp'`

3. Use the `--quiet` flag with `nextflow run` to suppress the printing of each file that is detected by the pipeline.

9.1.2 Configuration

For each sample, this pipeline take a bam and a fragments file. These can be specified separately, or from a Cell Ranger ATAC/ARC `outs/` path.

Input with independent bam and fragments files

Use the profiles `bam` and `fragments`:

```
nextflow config vib-singlecell-nf/vsn-pipelines/main_atac.nf \
    -profile atac_qc_filtering,bam,fragments,vsc,pycistopic_hg38 \
    > atac_qc.config
```

Preset profiles are available for human (`pycistopic_hg38`), mouse (`pycistopic_mm10`), and fly (`pycistopic_dmel`). Or, these profiles can be omitted and set manually in the config (biomart, macs2).

Input data (bam and fragments files) are specified in the data section:

```
data {
   fragments {
      file_paths = '/staging/leuven/stg_00002/lcb/cflerin/analysis/asap/20210527_
↪hydrop-atac_asabr/atac_preprocess/out_run1/data/fragments/ASA__*tsv.gz'
      suffix = '.sinto.fragments.tsv.gz'
      index_extension = '.tbi'
   }
   bam {
      file_paths = '/staging/leuven/stg_00002/lcb/cflerin/analysis/asap/20210527_
↪hydrop-atac_asabr/atac_preprocess/out_run1/data/bam/ASA*bam'
      suffix = '.bwa.out.possorted.bam'
      index_extension = '.bai'
   }
}
```

Multiple files can be specified with `*` in `file_paths` or by separating the paths with a comma.

Warning: The `suffix` for both bam and fragments will be removed from the filename to get sample IDs. The sample names obtained must match between bam and fragments for the files to be paired properly in the workflow.

Input with Cell Ranger ATAC data

Use the `tenx_atac` profile:

```
nextflow config vib-singlecell-nf/vsn-pipelines/main_atac.nf \
    -profile atac_qc_filtering,tenx_atac,vsc,pycistopic_hg38 \
    > atac_qc.config
```

Input data (the Cell Ranger `outs/` path) are specified in the data section:

```
data {
    tenx_atac {
        cellranger_mex = '/data/cellranger_atac_2.0/*/outs,/data/processed/cellranger_
↪arc_2.0.0/*/outs'
    }
}
```

Multiple files can be specified with * in `tenx_atac` or by separating the paths with a comma.

Input directly from the preprocessing pipeline

It is also possible to run these QC steps directly after the `atac_preprocess` pipeline, with a single command. In this case, all the appropriate configuration profiles must be included at the configuration start:

```
nextflow config vib-singlecell-nf/vsn-pipelines/main_atac.nf \
    -profile atac_preprocess,atac_qc_filtering,pycistopic_hg38,vsc \
    > atac_preprocess_and_qc.config
```

Note that here, we do not include `bam` and `fragments` profiles that specify the input data locations to the QC steps since these are piped directly from the preprocessing pipeline. One caveat to this is that it could potentially make it harder to run the qc pipeline with `-resume` later on, especially if the Nextflow `work/` directory is not saved due to disk space concerns.

To execute the preprocessing and mapping pipeline in one step, use the `atac_preprocess_with_qc` entry point:

```
nextflow -C atac_preprocess_and_qc.config run \
    vib-singlecell-nf/vsn-pipelines/main_atac.nf \
    -entry atac_preprocess_with_qc -resume --quiet
```

9.1.3 Execution

After configuring, the workflow can be run with:

```
nextflow -C atac_qc.config run \
    vib-singlecell-nf/vsn-pipelines/main_atac.nf \
    -entry atac_qc_filtering --quiet -resume
```

After completing, view the report in `out/notebooks/<project_name>__pycisTopic_QC_report.html`. To change the filtering settings, use the `params.tools.pycistopic.call_cells` section.

9.1.4 Adjusting the filter settings

In the pycisTopic parameters, filter settings can be applied in this section:

```
pycistopic {
    call_cells {
        report_ipynb = '/src/pycistopic/bin/pycisTopic_qc_report_template.ipynb'
        use_density_coloring_on_scatterplot = true
        use_detailed_title_on_scatterplot = true
        filter_frags_lower = '1000'
        filter_frags_upper = ''
        filter_tss_lower = '8'
        filter_tss_upper = ''
        filter_frip_lower = ''
        filter_frip_upper = ''
        filter_dup_rate_lower = ''
        filter_dup_rate_upper = ''
    }
}
```

If a setting is empty (`''`), this filter will not be applied. If set to a single value (i.e. `filter_frags_lower=1000`), this will apply this filter value to all samples. To use sample-specific filters, this can be written as:

```
filter_frags_lower = [
  'default': 1000,
  'Sample_1': 1500,
  'Sample_2': 2000,
]
```

The `default` setting (optional) is applied to all samples not listed in array. If this default setting is missing, no filter will be applied to samples not listed in the array (all barcodes kept).

After setting the filters, the pipeline can be re-run to apply the new filters (use `-resume`).

The additional settings control the output of the scatter plots in the report: * `use_density_coloring_on_scatterplot`: Slower when turned on; it can be helpful to set this to `false` until the proper thresholds are determined. * `use_detailed_title_on_scatterplot`: Adds the cell count and median values after filtering to the title of each plot.

9.2 Output

An example output tree is shown here.

```
out/
├── data
│   ├── macs2
│   │   ├── sample_1.peaks.narrowPeak
│   │   ├── sample_1.summits.bed
│   │   ├── sample_2.peaks.narrowPeak
│   │   └── sample_2.summits.bed
│   └── pycistopic
│       └── qc
│           ├── benchmark_library_downsampled__metadata.pickle
│           ├── benchmark_library_downsampled__profile_data.pickle
│           ├── selected_barcodes
│           │   ├── sample_1.cell_barcodes.txt
│           │   └── sample_2.cell_barcodes.txt
│           └── selected_barcodes_nFrag
│               ├── sample_1.barcodes_nFrag_thr.txt
│               └── sample_2.barcodes_nFrag_thr.txt
└── notebooks
    ├── example_project__pycisTopic_QC_report.html
    └── example_project__pycisTopic_QC_report.ipynb
```

- `macs2`: contains the narrowPeak and bed file for each sample.
- `pycistopic`: * `qc`: contains Python objects (in pickle format) for the metadata and profile data computed by pycisTopic.
  - `selected_barcodes`: contains a text file with selected cell barcodes (one per line) based on the thresholds set in the config file.
  - `selected_barcodes_nFrag`: contains a text file with barcodes (one per line) that have unique fragment counts greater than the `params.tools.pycistopic.compute_qc_stats.n_frag` setting in the pycisTopic parameters.

CHAPTER 10

Development Guide

10.1 Create module

Tool-based modules are located in `src/<tool-name>`, and each module has a specific structure for scripts and Nextflow processes (see *Repository structure* below).

10.1.1 Case study: Add *Harmony*

Harmony is a method published in Nature Methods that performs integration of single-cell data.

Links:

- GitHub: https://github.com/immunogenomics/harmony
- Tutorial: https://github.com/immunogenomics/harmony/blob/master/vignettes/quickstart.Rmd

Steps:

1. Create a new issue on `vsn-pipelines` GitHub repository explaining which module you are going to add (e.g.: *Add Harmony batch correction method*).
2. Fork the `vsn-pipelines` repository to your own GitHub account (if you are an external collaborator).
3. From your local copy of `vsn-pipelines` GitHub repository, create a new branch called `feature/[github-issue-id]-[description]`.

In this case,

- `[github-issue-id] = 115`
- `[description] = add_harmony_batch_correction_method`

It is highly recommended to start from the `develop` branch:

```
git checkout develop
git fetch
git pull
git checkout -b feature/115-add_harmony_batch_correction_method
```

1. Use the template repository in the `vib-singlecell-nf` organisation to create the framework for the new module in `src/<tool-name>`:

```
git clone --depth=1 https://github.com/vib-singlecell-nf/template.git src/
↪harmony
```

2. Now, you can start to edit file in the tool module that is now located in `src/<tool-name>`. Optionally, you can delete the `.git` directory in the new module to avoid confusion in future local development:

```
rm -rf src/harmony/.git
```

3. Create the Dockerfile recipe

```
FROM continuumio/miniconda3

SHELL ["/bin/bash", "--login", "-c"]

ADD environment.yml /tmp/environment.yml
RUN conda env create -f /tmp/environment.yml

RUN head -1 /tmp/environment.yml | cut -d' ' -f2 > /tmp/version \
&& ln -s "/opt/conda/envs/$(cat /tmp/version)" /opt/conda/venv

# Initialize conda in bash config files:
RUN conda init bash

# Activate the environment, and make sure it's activated:
RUN echo "conda activate $(cat /tmp/version)" >> ~/.bashrc && \
conda activate $(cat /tmp/version) && \
R -e "devtools::install_github(repo = 'dynverse/anndata', ref = '0.7.5.2')
↪" && \
R -e "devtools::install_github(repo = 'aertslab/SCopeLoomR')"

RUN apt-get -y update \
# Need to run ps
&& apt-get -y install procps \
&& apt-get -y install libxml2 \
# Clean
&& conda clean -afy \
&& rm -rf /var/cache/apt/* \
&& rm -rf /var/lib/apt/lists/*

RUN echo "source activate $(cat /tmp/version)" >> ~/.bashrc
ENV PATH="/opt/conda/venv/bin:${PATH}"
```

```
# environment.yml
name: harmony-v1.0-3
channels:
- r
- conda-forge
- bioconda
dependencies:
```

(continues on next page)

(continued from previous page)

```
- python=3.7
- anndata=0.7.6
- r-base=4.0.2
- r-argparse=2.0.1
- r-devtools
- r-reticulate=1.20
- r-hdf5r
- r-harmony
```

4. Rename the `nextflow.config` file to create the `harmony.config` configuration file.

    - Each process's options should be in their own level. With a single process, you do not need one extra level. The `report_ipynb` Jupyter Notebook is available here.

```
params {
    tools {
        harmony {
            container = 'vibsinglecellnf/harmony:1.0'
            report_ipynb = "${params.misc.test.enabled ? '../../..' : ''}/
↪src/harmony/bin/reports/sc_harmony_report.ipynb"
            varsUse = ['batch']
        }
    }
}
```

5. Create the R script to run Harmony

```
print("################################################################")
print("# Harmony: Algorithm for single cell integration               #")
print('# GitHub: https://github.com/immunogenomics/harmony            #')
print('# Paper: https://www.nature.com/articles/s41592-019-0619-0     #')
print("################################################################")

# Loading dependencies scripts
library("argparse")
library("reticulate")
library("anndata")

# Link Python to this R session
use_python("/opt/conda/envs/harmony-v1.0-3/bin")
Sys.setenv(RETICULATE_PYTHON = "/opt/conda/envs/harmony-v1.0-3/bin")

parser <- ArgumentParser(description='Scalable integration of single cell␣
↪RNAseq data for batch correction and meta analysis')
parser$add_argument(
    'input',
    metavar='INPUT',
    type="character",
    help='Input file [default]'
)
parser$add_argument(
    '--output-prefix',
    type="character",
    dest='output_prefix',
    default = "foo",
    help="Prefix path to save output files. [default %default]"
)
```

(continues on next page)

(continued from previous page)

```
parser$add_argument(
    '--seed',
    type="character",
    dest='seed',
    default=617,
    help='Seed. [default %default]'
)
parser$add_argument(
    "--vars-use",
    type="character",
    dest='vars_use',
    action="append",
    default=NULL,
    help='If meta_data is dataframe, this defined which variable(s) to␣
↪remove (character vector).'
)
parser$add_argument(
    '--do-pca',
    type="logical",
    dest='do_pca',
    action="store",
    default=FALSE,
    help='Whether to perform PCA on input matrix.'
)
parser$add_argument(
    '--theta',
    type="double",
    dest='theta',
    default=NULL,
    help='Diversity clustering penalty parameter. Specify for each␣
↪variable in vars_use Default theta=2. theta=0 does not encourage any␣
↪diversity. Larger values of theta result in more diverse clusters.␣
↪[default %default]'
)
parser$add_argument(
    '--lambda',
    type="double",
    dest='lambda',
    default=NULL,
    help='Ridge regression penalty parameter. Specify for each variable␣
↪in vars_use. Default lambda=1. Lambda must be strictly positive.␣
↪Smaller values result in more aggressive correction. [default %default]'
)
parser$add_argument(
    '--epsilon-harmony',
    type="double",
    dest='epsilon_harmony',
    default=1e-04,
    help='Convergence tolerance for Harmony. Set to -Inf to never stop␣
↪early. [default %default]'
)


args <- parser$parse_args()

if(args$epsilon_harmony < 0) {
    args$epsilon_harmony <- -Inf
```

(continues on next page)

(continued from previous page)

```
    print("Setting epsilon.harmony argument to -Inf...")
}

cat("Parameters: \n")
print(args)

if(is.null(args$vars_use)) {
    stop("The parameter --vars-use has to be set.")
}

# Required by irlba::irlba (which harmony depends on) for reproducibility
if(!is.null(args$seed)) {
    set.seed(args$seed)
} else {
    warnings("No seed is set, this will likely give none reproducible␣
↪results.")
}

# Required for reproducibility in case numeric parameters are passed (e.g.
↪: theta, lambda)
args <- lapply(X = args, FUN = function(arg) {
    if(is.numeric(x = arg)) {
        if(arg %% 1 == 0) {
            return (as.integer(x = arg))
        } else {
            return (arg)
        }
    }
    return (arg)
})

input_ext <- tools::file_ext(args$input)

if(input_ext == "h5ad") {
    adata <- anndata::read_h5ad(filename = args$input)
    if(!("X_pca" %in% names(x = adata$obsm))) {
        stop("X_pca slot is not found in the AnnData (h5ad).")
    }
    obs <- adata$obs
    pca_embeddings <- adata$obsm[["X_pca"]]
    row.names(x = pca_embeddings) <- row.names(x = obs)
    colnames(x = pca_embeddings) <- paste0("PCA_", seq(from = 1, to =␣
↪ncol(x = pca_embeddings)))
    metadata <- obs
} else {
    stop(paste0("Unrecognized input file format: ", input_ext, "."))
}

print(paste0("PCA embeddings matrix has ", dim(x = pca_embeddings)[1], "␣
↪rows, ", dim(x = pca_embeddings)[2], " columns."))

if(sum(args$vars_use %in% colnames(x = metadata)) != length(x = args$vars_
↪use)) {
    stop("Some argument value from the parameter(s) --vars-use are not␣
↪found in the metadata.")
}
```

(continues on next page)

(continued from previous page)

```
print(paste0("Batch variables used for integration: ", paste0(args$vars_
↪use, collapse=", ")))

# Run Harmony
# Expects PCA matrix (Cells as rows and PCs as columns.)
harmony_embeddings <- harmony::HarmonyMatrix(
    data_mat = pca_embeddings,
    meta_data = metadata,
    vars_use = args$vars_use,
    do_pca = args$do_pca,
    theta = args$theta,
    lambda = args$lambda,
    epsilon.harmony = args$epsilon_harmony,
    verbose = FALSE
)

# Save the results
## PCA corrected embeddings
write.table(
    x = harmony_embeddings,
    file = paste0(args$output_prefix, ".tsv"),
    quote = FALSE,
    sep = "\t",
    row.names = TRUE,
    col.names = NA
)
```

6. Create the Nextflow process that will run the Harmony R script defined in the previous step.

```
nextflow.preview.dsl=2

binDir = !params.containsKey("test") ? "${workflow.projectDir}/src/
↪harmony/bin/" : ""

process SC__HARMONY__HARMONY_MATRIX {

    container params.tools.harmony.container
    publishDir "${params.global.outdir}/data/intermediate", mode: 'symlink
↪'
    label 'compute_resources__default'

    input:
        tuple \
            val(sampleId), \
            path(f)

    output:
        tuple \
            val(sampleId), \
            path("${sampleId}.SC__HARMONY__HARMONY_MATRIX.tsv")

    script:
        def sampleParams = params.parseConfig(sampleId, params.global,␣
↪params.tools.harmony)
        processParams = sampleParams.local
        varsUseAsArguments = processParams.varsUse.collect({ '--vars-use'␣
↪+ ' ' + it }).join(' ')
```

(continues on next page)

(continued from previous page)

```
        """
        ${binDir}run_harmony.R \
            ${f} \
            --seed ${params.global.seed} \
            ${varsUseAsArguments} \
            ${processParams?.theta ? "--theta "+ processParams.theta : ""
↪} \
            ${processParams?.lambda ? "--lambda "+ processParams.lambda :
↪"" } \
            ${processParams?.epsilonHarmony ? "--epsilon-harmony "+
↪processParams.epsilonHarmony : "" } \
            --output-prefix "${sampleId}.SC__HARMONY__HARMONY_MATRIX"
        """

}
```

7. Create a Nextflow "subworkflow" that will call the Nextflow process defined in the previous step and perform some other tasks (dimensionality reduction, cluster identification, marker genes identification and report generation). This step is not required. However it is skipped, the code would still need to added into the main `harmony` workflow (*workflows/harmony.nf*, see the next step)

```
nextflow.preview.dsl=2

////////////////////////////////////////////////////
//  process imports:

include {
    clean;
} from '../../utils/processes/utils.nf' params(params)
include {
    COMBINE_BY_PARAMS;
} from "../../utils/workflows/utils.nf" params(params)
include {
    PUBLISH as PUBLISH_BEC_OUTPUT;
    PUBLISH as PUBLISH_BEC_DIMRED_OUTPUT;
    PUBLISH as PUBLISH_FINAL_HARMONY_OUTPUT;
} from "../../utils/workflows/utils.nf" params(params)

include {
    SC__HARMONY__HARMONY_MATRIX;
} from './../processes/runHarmony.nf' params(params)
include {
    SC__H5AD_UPDATE_X_PCA;
} from './../../utils/processes/h5adUpdate.nf' params(params)
include {
    NEIGHBORHOOD_GRAPH;
} from './../../scanpy/workflows/neighborhood_graph.nf' params(params)
include {
    DIM_REDUCTION_TSNE_UMAP;
} from './../../scanpy/workflows/dim_reduction.nf' params(params)
include {
    SC__SCANPY__CLUSTERING_PARAMS;
} from './../../scanpy/processes/cluster.nf' params(params)
include {
    CLUSTER_IDENTIFICATION;
} from './../../scanpy/workflows/cluster_identification.nf'
↪params(params) // Don't only import a specific process (the function
↪needs also to be imported)
```

(continues on next page)

(continued from previous page)

```
// reporting:
include {
    GENERATE_DUAL_INPUT_REPORT
} from './../../scanpy/workflows/create_report.nf' params(params)

////////////////////////////////////////////////////
//  Define the workflow

workflow BEC_HARMONY {

    take:
        normalizedTransformedData
        dimReductionData
        // Expects (sampleId, anndata)
        clusterIdentificationPreBatchEffectCorrection

    main:
        // Run Harmony
        harmony_embeddings = SC__HARMONY__HARMONY_MATRIX(
            dimReductionData.map {
                it -> tuple(it[0], it[1])
            }
        )
        SC__H5AD_UPDATE_X_PCA(
            dimReductionData.map {
                it -> tuple(it[0], it[1])
            }.join(harmony_embeddings)
        )

        PUBLISH_BEC_OUTPUT(
            SC__H5AD_UPDATE_X_PCA.out.map {
                // if stashedParams not there, just put null 3rd arg
                it -> tuple(it[0], it[1], it.size() > 2 ? it[2]: null)
            },
            "BEC_HARMONY.output",
            "h5ad",
            null,
            false
        )

        NEIGHBORHOOD_GRAPH(
            SC__H5AD_UPDATE_X_PCA.out.join(
                dimReductionData.map {
                    it -> tuple(it[0], it[2], *it[3..(it.size()-1)])
                }
            )
        )

        // Run dimensionality reduction
        DIM_REDUCTION_TSNE_UMAP( NEIGHBORHOOD_GRAPH.out )

        PUBLISH_BEC_DIMRED_OUTPUT(
            DIM_REDUCTION_TSNE_UMAP.out.dimred_tsne_umap,
            "BEC_HARMONY.dimred_output",
            "h5ad",
            null,
```

(continues on next page)

(continued from previous page)

```
            false
        )

        // Run clustering
        // Define the parameters for clustering
        def clusteringParams = SC__SCANPY__CLUSTERING_PARAMS(␣
↪clean(params.tools.scanpy.clustering) )
        CLUSTER_IDENTIFICATION(
            normalizedTransformedData,
            DIM_REDUCTION_TSNE_UMAP.out.dimred_tsne_umap,
            "Post Batch Effect Correction (Harmony)"
        )

        marker_genes = CLUSTER_IDENTIFICATION.out.marker_genes.map {
            it -> tuple(
                it[0], // sampleId
                it[1], // data
                !clusteringParams.isParameterExplorationModeOn() ? null :␣
↪it[2..(it.size()-1)], // Stash params
            )
        }

        PUBLISH_FINAL_HARMONY_OUTPUT(
            marker_genes.map {
                it -> tuple(it[0], it[1], it[2])
            },
            "BEC_HARMONY.final_output",
            "h5ad",
            null,
            clusteringParams.isParameterExplorationModeOn()
        )

        // This will generate a dual report with results from
        // - Pre batch effect correction
        // - Post batch effect correction
        becDualDataPrePost = COMBINE_BY_PARAMS(
            clusterIdentificationPreBatchEffectCorrection,
            // Use PUBLISH output to avoid "input file name collision"
            PUBLISH_FINAL_HARMONY_OUTPUT.out,
            clusteringParams
        )
        harmony_report = GENERATE_DUAL_INPUT_REPORT(
            becDualDataPrePost,
            file(workflow.projectDir + params.tools.harmony.report_ipynb),
            "SC_BEC_HARMONY_report",
            clusteringParams.isParameterExplorationModeOn()
        )

    emit:
        data = CLUSTER_IDENTIFICATION.out.marker_genes
        cluster_report = CLUSTER_IDENTIFICATION.out.report
        harmony_report

}
```

8. In the `vsn-pipelines`, create a new main workflow called `harmony.nf` under `workflows/`

```
nextflow.enable.dsl=2

////////////////////////////////////////////////////
//  Import sub-workflows/processes from the utils module:
include {
    getBaseName
} from '../src/utils/processes/files.nf'
include {
    clean;
    SC__FILE_CONVERTER;
    SC__FILE_CONCATENATOR;
} from '../src/utils/processes/utils.nf' params(params)
include {
    COMBINE_BY_PARAMS;
} from '../src/utils/workflows/utils.nf' params(params)
include {
    FINALIZE;
} from '../src/utils/workflows/finalize.nf' params(params)
include {
    FILTER_AND_ANNOTATE_AND_CLEAN;
} from '../src/utils/workflows/filterAnnotateClean.nf' params(params)
include {
    UTILS__GENERATE_WORKFLOW_CONFIG_REPORT;
} from '../src/utils/processes/reports.nf' params(params)

////////////////////////////////////////////////////
//  Import sub-workflows/processes from the tool module:
include {
    QC_FILTER;
} from '../src/scanpy/workflows/qc_filter.nf' params(params)
include {
    NORMALIZE_TRANSFORM;
} from '../src/scanpy/workflows/normalize_transform.nf' params(params)
include {
    HVG_SELECTION;
} from '../src/scanpy/workflows/hvg_selection.nf' params(params)
include {
    NEIGHBORHOOD_GRAPH;
} from '../src/scanpy/workflows/neighborhood_graph.nf' params(params)
include {
    DIM_REDUCTION_PCA;
} from '../src/scanpy/workflows/dim_reduction_pca.nf' params(params)
include {
    DIM_REDUCTION_TSNE_UMAP;
} from '../src/scanpy/workflows/dim_reduction.nf' params(params)
// cluster identification
include {
    SC__SCANPY__CLUSTERING_PARAMS;
} from '../src/scanpy/processes/cluster.nf' params(params)
include {
    CLUSTER_IDENTIFICATION;
} from '../src/scanpy/workflows/cluster_identification.nf' params(params)
include {
    BEC_HARMONY;
} from '../src/harmony/workflows/bec_harmony.nf' params(params)
include {
    SC__DIRECTS__SELECT_DEFAULT_CLUSTERING
```

(continues on next page)

(continued from previous page)

```
} from '../src/directs/processes/selectDefaultClustering.nf'
// reporting:
include {
    SC__SCANPY__MERGE_REPORTS;
} from '../src/scanpy/processes/reports.nf' params(params)
include {
    SC__SCANPY__REPORT_TO_HTML;
} from '../src/scanpy/processes/reports.nf' params(params)


workflow harmony {

    take:
        data

    main:
        // Data processing
        // To avoid variable 'params' already defined in the process scope
        def scanpyParams = params.tools.scanpy

        out = data | \
            SC__FILE_CONVERTER | \
            FILTER_AND_ANNOTATE_AND_CLEAN

        if(scanpyParams.containsKey("filter")) {
            out = QC_FILTER( out ).filtered // Remove concat
        }
        if(params.utils?.file_concatenator) {
            out = SC__FILE_CONCATENATOR(
                out.map {
                    it -> it[1]
                }.toSortedList(
                    { a, b -> getBaseName(a, "SC") <=> getBaseName(b, "SC
↪") }
                )
            )
        }
        if(scanpyParams.containsKey("data_transformation") &&␣
↪scanpyParams.containsKey("normalization")) {
            out = NORMALIZE_TRANSFORM( out )
        }
        out = HVG_SELECTION( out )
        DIM_REDUCTION_PCA( out.scaled )
        NEIGHBORHOOD_GRAPH( DIM_REDUCTION_PCA.out )
        DIM_REDUCTION_TSNE_UMAP( NEIGHBORHOOD_GRAPH.out )

        // Perform the clustering step w/o batch effect correction (for␣
↪comparison matter)
        clusterIdentificationPreBatchEffectCorrection = CLUSTER_
↪IDENTIFICATION(
            NORMALIZE_TRANSFORM.out,
            DIM_REDUCTION_TSNE_UMAP.out.dimred_tsne_umap,
            "Pre Batch Effect Correction"
        )

        // Perform the batch effect correction
        BEC_HARMONY(
```

(continues on next page)

(continued from previous page)

```
            NORMALIZE_TRANSFORM.out,
            // include only PCA since Harmony will correct this
            DIM_REDUCTION_PCA.out,
            clusterIdentificationPreBatchEffectCorrection.marker_genes
        )

        // Finalize
        FINALIZE(
            params.utils?.file_concatenator ? SC__FILE_CONCATENATOR.out :␣
↪SC__FILE_CONVERTER.out,
            BEC_HARMONY.out.data,
            'HARMONY.final_output'
        )

        // Define the parameters for clustering
        def clusteringParams = SC__SCANPY__CLUSTERING_PARAMS(␣
↪clean(scanpyParams.clustering) )

        // Select a default clustering when in parameter exploration mode
        if(params.tools?.directs && clusteringParams.
↪isParameterExplorationModeOn()) {
            scopeloom = SC__DIRECTS__SELECT_DEFAULT_CLUSTERING( FINALIZE.
↪out.scopeloom )
        } else {
            scopeloom = FINALIZE.out.scopeloom
        }

        // Reporting

        project = CLUSTER_IDENTIFICATION.out.marker_genes.map { it ->␣
↪it[0] }
        UTILS__GENERATE_WORKFLOW_CONFIG_REPORT(
            file(workflow.projectDir + params.utils.workflow_
↪configuration.report_ipynb)
        )

        // Collect the reports:
        // Pairing clustering reports with bec reports
        if(!clusteringParams.isParameterExplorationModeOn()) {
            clusteringBECReports = BEC_HARMONY.out.cluster_report.map {
                it -> tuple(it[0], it[1])
            }.combine(
                BEC_HARMONY.out.harmony_report.map {
                    it -> tuple(it[0], it[1])
                },
                by: 0
            ).map {
                it -> tuple(it[0], *it[1..it.size()-1], null)
            }
        } else {
            clusteringBECReports = COMBINE_BY_PARAMS(
                BEC_HARMONY.out.cluster_report.map {
                    it -> tuple(it[0], it[1], *it[2])
                },
                BEC_HARMONY.out.harmony_report,
                clusteringParams
            )
```

(continues on next page)

(continued from previous page)

```
        }
        ipynbs = project.combine(
            UTILS__GENERATE_WORKFLOW_CONFIG_REPORT.out
        ).join(
            HVG_SELECTION.out.report.map {
                it -> tuple(it[0], it[1])
            }
        ).combine(
            clusteringBECReports,
            by: 0
        ).map {
            it -> tuple(it[0], it[1..it.size()-2], it[it.size()-1])
        }

        SC__SCANPY__MERGE_REPORTS(
            ipynbs,
            "merged_report",
            clusteringParams.isParameterExplorationModeOn()
        )
        SC__SCANPY__REPORT_TO_HTML(SC__SCANPY__MERGE_REPORTS.out)

    emit:
        filteredloom = FINALIZE.out.filteredloom
        scopeloom = scopeloom
        scanpyh5ad = FINALIZE.out.scanpyh5ad

}
```

9. Add a new Nextflow profile in the `profiles` section of the main `nextflow.config` of the `vsn-pipelines` repository:

```
profiles {

    harmony {
        includeConfig 'src/scanpy/scanpy.config'
        includeConfig 'src/harmony/harmony.config'
    }
    ...
}
```

10. Finally add a new entry in `main.nf` of the `vsn-pipelines` repository.

```
// run multi-sample with bbknn, output a scope loom file
workflow harmony {

    include {
        harmony as HARMONY
    } from './workflows/harmony' params(params)
    include {
        PUBLISH as PUBLISH_HARMONY
    } from "./src/utils/workflows/utils" params(params)

    getDataChannel | HARMONY
    PUBLISH_HARMONY(
        HARMONY.out.scopeloom,
        "HARMONY",
```

(continues on next page)

(continued from previous page)

```
        "loom",
        null,
        false
    )

}
```

11. You should now be able to configure (`nextflow config ...`) and run the `harmony` pipeline (`nextflow run ...`).
12. After confirming that your module is functional, you should create a pull request to merge your changes into the `develop` branch.
    - Make sure you have removed all references to `TEMPLATE` in your repository
    - Include some basic documentation for your module so people know what it does and how to use it.

The pull request will be reviewed and accepted once it is confirmed to be working. Once the `develop` branch is merged into `master`, the new tool will be part of the new release of VSN Pipelines!

10.2 Repository structure

10.2.1 Root

The repository root contains a `main.nf` and associated `nextflow.config`. The root `main.nf` imports and calls sub-workflows defined in the modules.

10.2.2 Modules

A "module" consists of a folder labeled with the tool name (Scanpy, SCENIC, utils, etc.), with subfolders for

- `bin/` (scripts passed into the container)
- `processes/` (where Nextflow processes are defined)

The root of the modules folder contains workflow files + associated configs (as many as there are workflows):

- `main.nf` + `nextflow.config`
- `single_sample.nf` + `scenic.config`
- ...

```
src/
├── cellranger
│   ├── main.nf
│   ├── nextflow.config
│   └── processes
│       ├── count.nf
│       └── mkfastq.nf
│
├── channels
│   └── tenx.nf
│
├── scenic
│   ├── bin
```

(continues on next page)

(continued from previous page)

```
│   │   ├── grnboost2_without_dask.py
│   ├── processes
│   │   ├── aucell.nf
│   │   ├── cistarget.nf
│   │   ├── grnboost2withoutDask.nf
│   ├── main.nf
│   └── scenic.config
│
└── utils
    ├── bin
    │   ├── h5ad_to_loom.py
    │   ├── sc_file_concatenator.py
    │   └── sc_file_converter.py
    ├── utils.config
    └── processes
        ├── files.nf
        ├── h5ad_to_loom.nf
        ├── utils_1.test.nf
        ├── utils_2.test.nf
        └── utils.nf
```

10.2.3 Workflows

Workflows (chains of nf processes) are defined in the module root folder (e.g. src/Scanpy/bec_bbknn.nf) Workflows import multiple processes and define the workflow by name:

```
include SC__CELLRANGER__MKFASTQ from './processes/mkfastq'  params(params)
include SC__CELLRANGER__COUNT   from './processes/count'    params(params)

workflow CELLRANGER {

    main:
        SC__CELLRANGER__MKFASTQ(file(params.tools.cellranger.mkfastq.csv),␣
↪path(params.tools.cellranger.mkfastq.runFolder))
        SC__CELLRANGER__COUNT(file(params.tools.cellranger.count.transcriptome), SC__
↪CELLRANGER__MKFASTQ.out.flatten())
    emit:
        SC__CELLRANGER__COUNT.out

}
```

10.2.4 Workflow imports

Entire **sub-workflows** can also be imported in other workflows with one command (inheriting all of the process imports from the workflow definition):

```
include CELLRANGER from '../cellranger/main.nf' params(params)
```

This leads to the ability to easily define **high-level workflows** in the master Nextflow file: `vib-singlecell-nf/vsn-pipelines/main.nf`:

```
include CELLRANGER from './src/cellranger/main.nf' params(params)
include BEC_BBKNN from './src/scanpy/bec_bbknn.nf' params(params)
```

(continues on next page)

(continued from previous page)

```
include SCENIC from './src/scenic/main.nf' params(params)

workflow {

    CELLRANGER()
    BEC_BBKNN( CELLRANGER.out )
    SCENIC( BEC_BBKNN.out )

}
```

10.2.5 Parameters structure

Parameters are stored in a separate config file per workflow, plus the main `nextflow.config`. These parameters are merged when starting the run using e.g.:

```
includeConfig 'src/scenic/nextflow.config'
```

The parameter structure internally (post-merge) is:

```
params {
    global {
        baseFilePath = "/opt/vib-singlecell-nf"
        project_name = "MCF7"
        ...
    }
    tools {
        utils {
            file_converter {
                ...
            }
            file_annotator {
                ...
            }
            file_concatenator {
                ...
            }
        }
        scanpy {
            container = 'docker://vib-singlecell-nf/scanpy:1.8.1'
            filter {
                ...
            }
            data_transformation {
                ...
            }
            normalization {
                ...
            }
            feature_selection {
                ...
            }
            feature_scaling {
                ...
            }
            dim_reduction {
```

(continues on next page)

(continued from previous page)

```
                pca {
                    method = 'pca'
                    ...
                }
                umap {
                    method = 'tsne'
                    ...
                }
            }
            batch_effect_correct {
                ...
            }
            clustering {
                ...
            }
        }
    }
}
```

10.3 Module testing

Modules and processes can be tested independently, you can find an example in `src/utils/main.test.nf`.

The `SC__FILE_CONVERTER` process is tested against the `tiny` dataset available in `data/01.count`.

CHAPTER 11

Attributions

VSN-Pipelines is a collection of workflows targeted toward the analysis of single cell data. VSN is dependendent on, and takes functions from many tools, developed both internally and externally, which are listed here.

11.1 Tools

- GreenleafLab/ArchR
- caleblareau/bap
- lh3/bwa
- Samtools
- campbio/celda
- Directs
- DropletUtils
- Drop-seq Tools
- EDirect
- OpenGene/fastp
- hangnoh/flybaseR
- dweemx/flybaseR
- immunogenomics/harmony
- pcacv
- Picard
- statgen/popscle
- aertslab/popscle_helper_tools

- aertslab/cisTopic
- theislab/scanpy
- aertslab/pySCENIC
- aertslab/SCENIC
- swolock/scrublet
- aertslab/single_cell_toolkit
- timoast/sinto
- constantAmateur/SoupX
- ncbi/sra-tools
- alexdobin/STAR
- Trim Galore

CHAPTER 12

VSN-Pipelines

12.1 VSN-Pipelines has now been archived

2023-04-19 - Unfortunately due to lack of developers, VSN-pipelines is no longer being worked on and has been archived. The repo will remain in read-only mode from this point on.

A repository of pipelines for single-cell data analysis in Nextflow DSL2.

Full documentation is available on Read the Docs, or take a look at the Quick Start guide.

This main repo contains multiple workflows for analyzing single cell transcriptomics data, and depends on a number of tools, which are organized into subfolders within the `src/` directory. The VIB-Singlecell-NF organization contains this main repo along with a collection of example runs (VSN-Pipelines-examples). Currently available workflows are listed below.

If VSN-Pipelines is useful for your research, consider citing:

- VSN-Pipelines All Versions (latest): 10.5281/zenodo.3703108.

12.2 Raw Data Processing Workflows

These are set up to run Cell Ranger and DropSeq pipelines.

Table 1: Raw Data Processing Workflows

Pipeline / Entrypoint	Purpose	Documentation
cellranger	Process 10x Chromium data	cellranger
demuxlet_freemuxlet	Demultiplexing	demuxlet_freemuxlet
nemesh	Process Drop-seq data	nemesh

12.3 Single Sample Workflows

The **Single Sample Workflows** perform a "best practices" scRNA-seq analysis. Multiple samples can be run in parallel, treating each sample separately.

Table 2: Single Sample Workflows

Pipeline / Entrypoint	Purpose	Documentation
single_sample	Independent samples	
single_sample_scenic	Ind. samples + SCENIC	
scenic	SCENIC GRN inference	
scenic_multiruns	SCENIC run multiple times	
single_sample_scenic_multiruns	Ind. samples + multi-SCENIC	
single_sample_scrublet	Ind. samples + Scrublet	
decontx	DecontX	
single_sample_decontx	Ind. samples + DecontX	
single_sample_decontx_scrublet	Ind. samples + DecontX + Scrublet	

12.4 Sample Aggregation Workflows

Sample Aggregation Workflows: perform a "best practices" scRNA-seq analysis on a merged and batch-corrected group of samples. Available batch correction methods include BBKNN, mnnCorrect, and Harmony.

Table 3: Sample Aggregation Pipelines

Pipeline / Entrypoint	Purpose	Documentation
bbknn	Sample aggregation + BBKNN	
bbknn_scenic	BBKNN + SCENIC	
harmony	Sample aggregation + Harmony	
harmony_scenic	Harmony + SCENIC	
mnncorrect	Sample aggregation + mnnCorrect	

In addition, the pySCENIC implementation of the SCENIC workflow is integrated here and can be run in conjunction with any of the above workflows. The output of each of the main workflows is a loom-format file, which is ready for import into the interactive single-cell web visualization tool SCope. In addition, data is also output in h5ad format, and reports are generated for the major pipeline steps.

12.5 scATAC-seq workflows

Single cell ATAC-seq processing steps are now included in VSN Pipelines. Currently, a preprocesing workflow is available, which will take fastq inputs, apply barcode correction, read trimming, bwa mapping, and output bam and fragments files for further downstream analysis. See here for complete documentation.