SalusSTS Analysis Workflow User Manual

Results

Logs

2025/7/28

Overview

SalusSTS is an analysis workflow designed for Salus high-resolution spatial transcriptomics capture chips. It enables users to analyze whole transcriptomes in both fresh frozen and paraffin-embedded tissue sections. SalusSTS includes quality control, alignment to match reads, generation of expression matrices, automatic segmentation of tissue regions, clustering, and other extended analyses.

1. Getting Started

1.1 System Requirements

SalusSTS runs on Linux systems with the following minimum requirements:

- 8-core processor (32 cores recommended)
- 128GB RAM (256GB recommended)
- 1TB storage space
- 64-bit CentOS/RedHat 7.8 or Ubuntu 20.04 and above

To run the workflow on a server cluster, the following conditions must be met:

- Shared file system (e.g., NFS)
- Slurm workload management system

1.2 Software Dependencies

Singularity

1.3 Download and Install

Step 1: Install Go and Singularity

Singularity v3 and above are primarily developed using the Go programming language, so you need to install Go to compile the source code. It is recommended to use Go v1.20.4 (https://go.dev/dl/) and Singularity v3.10.4 (https://github.com/sylabs/singularity).

Step 2: Download the SalusSTS image

Domestic storage path: The file shared via cloud drive: salusts.v1.0.sif

Link: https://pan.baidu.com/s/14SBKmZEF07Dz21XdLq-3jg Extraction code: nxhw

International storage path:

You can download and place the image file anywhere that is convenient.

2 Inputs

2.1 Build References Index

The Salus analysis workflow is compatible with 10X Genomics pre-built reference genomes. For users building a custom reference genome index, please refer to the STAR software tutorial. You will need to download the reference genome files for your species (fasta, gtf). Assuming the file tree is as follows:

Shell

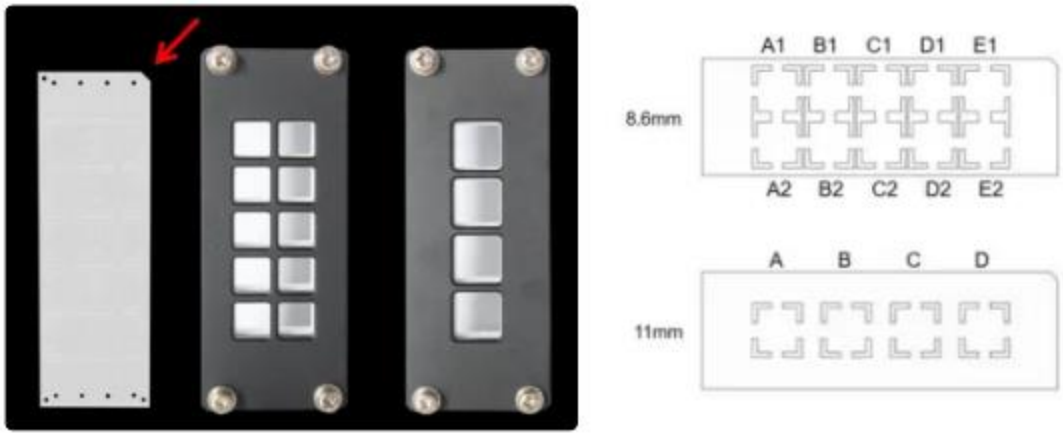
```
/share/data/reference/rat/
  └── ensembl110
      ├── Rattus_norvegicus.fa
      ├── Rattus_norvegicus.gtf
      └── Ratrefindex
/share/home/salus/
  └── salusts.v1.0.sif
```

Before running Singularity, you need to map the internal and external paths of the image. According to the file tree, /share is the common parent directory, so use the Singularity -B parameter to map /share. The command is as follows

```shell
singularity exec -B /share:/share --cleanenv \
/share/home/salus/salusts.v1.0.sif \
bash -c "export PATH=/opt/conda/bin/:$PATH &&
export LD_LIBRARY_PATH=/opt/conda/lib/:$LD_LIBRARY_PATH &&
STAR --runMode genomeGenerate \
--runThreadN 16 \
--genomeDir /share/data/reference/rat/ensembl110/Ratrefindex \
--genomeFastaFiles /share/data/reference/rat/ensembl110/Rattus_norvegicus.fa \
--sjdbGTFfile /share/data/reference/rat/ensembl110/Rattus_norvegicus.gtf \
--sjdbOverhang 99"
```

2.2 Slide types

SalusSTS currently supports two chip types: dual-row chips, which have a total of 10 valid wells, with each well measuring 8.6mm x 8.6mm; and single-row chips, which have 4 valid wells, with each well measuring 11mm x 11mm.



The Salus spatial transcriptomics chip features a notch in the upper right corner. For dual-row chips, the wells are named from bottom to top and left to right in the following order: A2, A1; B2, B1; C2, C1; D2, D1; E2, E1. Each experiment uses one well, and the Salus analysis workflow will crop and analyze data based on the specified well.

In the fstBC sequencing files, lane01 corresponds to all probe information for chip positions A1~E1, while lane02 corresponds to all probe information for chip positions A2~E2. For single-row chips, the wells are named from bottom to top in the order: A, B, C, D. In the fstBC sequencing files, lane01 contains all probe information for chip positions A~D.

Please note that in the config.yml file, single-row and dual-row chips correspond to different ChipType values.

2.3 Inputs for SalusSTS

To run the SalusSTS workflow, you must import three sets of fastq files: the fstBC file that records probe position information, and the user's sample library files R1 and R2. There are usually multiple pairs of R1 and R2 files.

fstBC: [Lane No.]_[Date]_[Machine No.]_BC.fastq.gz

For dual-row chips, Lane01 records the probe position information of the upper row, and Lane02 records the probe position information of the lower row; for single-row chips, the fstBC file only has files starting with Lane01.

R1: Records barcode and UMI sequence information in the user's sequencing data. The length is 40 nt, with the first 30 nt as the barcode and the last 10 nt as the UMI. Different sequencer settings may result in R1 files of varying lengths. If R1 contains a fixed intermediate sequence, config.yaml should be run in "cut" mode to trim it. If the barcode and UMI together form a 40-base sequence file, config.yaml should be run in "dark" mode.

R2: Records RNA insert fragment information from the user's sequencing data.

R1 and R2 file names must start with a letter, number, or underscore. File names can include letters, numbers, underscores, dots, hyphens, or spaces, but they must contain .R1 or .R2, _R1 or _R2 to identify read 1 or read 2. The extension can be any of fq.gz, fastq.gz, fq.gzip, or fastq.gzip. It is not required for file names to contain "Lane," nor is it necessary for files to be stored in subfolders corresponding to Lane*. Note that sometimes sequencing data obtained by users has already been pre-processed. R1 and R2 files have had adapters removed and quality filtering completed. fstBC files have undergone quality filtering and been converted to a new coordinate system. The workflow supports input of pre-processed sequencing data, and corresponding file naming and config.yaml settings will vary. Specific changes are recorded in the "Run with clean reads" section.

H&E stained images (optional): In spatial transcriptomics, H&E stained images can be combined with gene expression data to assist with tissue cutting, cell segmentation, and other tasks.

2.4 Configuration file Settings

Each time you run the Salus spatial transcriptomics SalusSTS analysis workflow, you need to create a new working directory, copy the config folder into this working directory, and make the necessary modifications. The config.yaml file is used to configure the program's parameters, while the fqlist.csv file records information about the input fastq files. Below is a demonstration of how to set up the configuration files for running a single sample or multiple samples on the same chip.

Shell

```
/share/home/salus/demo/workDir
  ├── config
  │ ├── config.yaml
  │ └── fqlist.csv
  └── HE
    ├── sampleID1.tif
    ├── sampleID2.tif
    └── sampleID3.tif
```

The completion of the config.yaml file is not affected by whether you are running a single sample or multiple samples. Below is an example of how to configure the config.yaml file. Please note:

1. When you first copy and paste the contents of config.yaml, you need to replace the separator before the "#" symbol with a space; 2. When entering each parameter, be sure to leave a space between the value and the ":" symbol.

config.yaml Shell

```yaml
fqs: config/fqlist.csv    # fastq file path

ChipType: Dual            # Solo, DualUpper, DualLower, Dual
                          # Solo corresponds to single-row chips; DualUpper corresponds to A1-E1, DualLower corresponds to A2-E2, Dual can handle A1-E1 and A2-E2 simultaneously.
ref: /share/data/reference/rat/ensembl110/Ratrefindex # Reference index path
SplitMask: 1              # split fstBC into 1-9 blocks to save memory usage
umilength: 10             # umi length
PEmode: dark              # Sequencing mode: cut or dark
BCUMI2pass: True          # Set True to allow 3 base mismatches, set False just allow 1 base
useSTARsort: False        # False for sorting by SAMTOOLS(less Mem and opened files required)
TissueSegment: True       # Set True to extract tissue region
TissueSegmentBinSize: 100 # Both automatic and manual segmentation require specifying pseudo_cell.binsize
HEtifPath: /share/home/salus/demo/workDir/HE          # HE path, can be anywhere
TissueSegmentHE: False    # For manual HE tissue segmentation step 1, read {HEtifPath}/{sampleID}.tif
ReadManualHEmask: False   # For manual HE tissue segmentation step 4, read mask file results/ManualHEmask/{sampleID}_{chipZone}.mask.tif
STAGATE: False            # Set True to use STAGATE, default is Flase
cleanedRead: False        # Start from reads after quality control
cuttedRead: False         # Start from reads after quality control and cut adapter, cleanRead must also be true
clean: True               # remove temporary file after all done, to save storage space
spliceJunction: None      # None means not output splice junction info, Standard means output splice junction info, default is None
Segment_anything: /share/data/sam_vit_h_4b8939.pth    # Path to SAM model
sndRNA:
 EnableSoloVelocyto: yes  # Set to yes if want cellbin result

pseudo_cell:
 binsize:                 # Users can specify any resolution
 - 400                    # It's about 100um x 100um
 - 100                    # It's about 25um x 25um (Forced generation)
 - 40                     # It's about 10um x 10um
```

Suppose the sequencing file storage path is as follows:

```shell
# fstBC
/share/data/sequence/chip/Lane01/
  ├── Lane01_0227_19B_BC.fastq.gz
  └── Lane02_0227_19B_BC.fastq.gz

# sample1
/share/data/sequence/libraryRat1/run1/
  ├── Lane01_0227_19B_D2_R1.fastq.gz
  └── Lane01_0227_19B_D2_R2.fastq.gz

/share/data/sequence/libraryRat1/run2/
  ├── Lane02_0312_18B_D2_R1.fastq.gz
  └── Lane02_0312_18B_D2_R2.fastq.gz

/share/data/sequence/libraryRat1/run3/
  ├── Lane03_0414_17B_D2_R1.fastq.gz
  └── Lane03_0414_17B_D2_R2.fastq.gz
# sample2
/share/data/sequence/libraryRat2/run1/
  ├── Lane04_0227_17B_A1_R1.fastq.gz
  └── Lane04_0227_17B_A1_R2.fastq.gz

/share/data/sequence/libraryRat2/run2/
  ├── Lane02_0312_17B_A1_R1.fastq.gz
  └── Lane02_0312_17B_A1_R2.fastq.gz

# sample3
/share/data/sequence/libraryRat3/
  ├── Lane01_0227_17B_B1_R1.fastq.gz
  └── Lane01_0227_17B_B1_R2.fastq.gz
```

Suppose the sequencing file storage path is as above:

Run1, run2, and run3 represent multiple sequencing runs of the same library. Below is an example of how to fill out the fqlist.csv file:

Run one sample

fqlist.csv Shell

```
seqType,fqPath,chipZone,sndBarcodePrimer,sampleID
fst,/share/data/sequence/chip/,,,X111
snd,/share/data/sequence/libraryRat1/run1/,D2,pe,Rat1
snd,/share/data/sequence/libraryRat1/run2/,D2,pe,Rat1
snd,/share/data/sequence/libraryRat1/run3/,D2,pe,Rat1
```

The following is the specification for filling out fqlist.csv. Please note that fqlist.csv must not contain extra blank lines, or an error will occur.

Parameters	Description
seqType	Used to indicate whether the input fastq sequence file of a chip (fst) or a user experimental sequencing file (snd); must be in lowercase.
fqPath	The sequencing files must be stored in the specified path and must have filenames ending with fastq.gz or fq.gz. By default, the snd row input path should contain both the R1 and R2 sequencing files.
chipZone	Record the well position used in the experiment; this will be used for cropping in subsequent analyses. Note that in fstBC sequencing files, this field does not need to be entered, and it must be in uppercase.
sndBarcodePrimer	Records the sequencing mode of the user's library after library preparation, either "pe" (paired-end) or "se" (single-end). This field does not need to be filled in for rows containing fstBC sequencing files and must be in lowercase
sampleID	The chip code corresponding to the fstBC row, and the user-defined sample name for the sndBC and sndNA rows. Please note that special characters such as "_" should be avoided in the sampleID name, as they will be treated as separators and may cause errors.

Run multiple samples

fqlist.csv Shell

```
seqType,fqPath,chipZone,sndBarcodePrimer,sampleID
fst,/share/data/sequence/chip/,,,X111
snd,/share/data/sequence/libraryRat1/run1/,D2,pe,Rat1
snd,/share/data/sequence/libraryRat1/run2/,D2,pe,Rat1
snd,/share/data/sequence/libraryRat1/run3/,D2,pe,Rat1
snd,/share/data/sequence/libraryRat2/run1/,A1,pe,Rat2
snd,/share/data/sequence/libraryRat2/run2/,A1,pe,Rat2
snd,/share/data/sequence/libraryRat3/,B1,pe,Rat3
```

The program does not support running files from different chips in the same working directory; only fstBC files from the same chip are allowed as input.

Run with clean reads

If the sequencing data you obtained has already had adapters removed and quality filtering completed, you need to set the **cleanedRead** and **cutedRead** parameters to **True** in the config.yaml file. Importing fstBC files that have not undergone quality filtering and coordinate system conversion in this workflow will result in abnormal outputs. All three input files must follow the new naming conventions:

- fstBC: /path1/[LaneNo.]/newCoord_fstBC_[sampleID]_[ChipType]_BC.fq.gz
- R1: /path2/[sampleID]_[chipZone]_R1.fq.gz
- R2: /path2/[sampleID]_[chipZone]_R2.fq.gz
- R1: /path3/[sampleID]_[chipZone]_R1.fq.gz
- R2: /path3/[sampleID]_[chipZone]_R2.fq.gz

Each pair of sndBC and sndNA files must be located within the same folder, and only one pair of fastq files is allowed per subfolder. There can be multiple folders under the directory, corresponding to different lanes. Note that each pair of fastq files is distinguished by the R1 and R2 suffixes, and their prefixes should be consistent and defined by the user. The **fqlist.csv** configuration setup can remain unchanged.

If the user receives high-quality sequencing data that has not had adapters removed, set **cleanedRead** to **True** and **cuttedRead** to **False**, and the file naming rules remain the same.

fqlist.csv Shell

```
seqType,fqPath,chipZone,sndBarcodePrimer,sampleID
fst,/share/data/sequence/chip/,,,X111
snd,/share/data/sequence/libraryRat1/run1/,D2,pe,Rat1
snd,/share/data/sequence/libraryRat1/run2/,D2,pe,Rat1
snd,/share/data/sequence/libraryRat1/run3/,D2,pe,Rat1
snd,/share/data/sequence/libraryRat2/run1/,A1,pe,Rat2
snd,/share/data/sequence/libraryRat2/run2/,A1,pe,Rat2
snd,/share/data/sequence/libraryRat3/,B1,pe,Rat3
```

3.Running Pipelines

Quick Start

The SalusSTS workflow has been packaged as an image, and users can easily start the workflow by following these steps:

Step 1. Enter the working directory for this analysis task, for example: /share/home/salus/demo/workDir.

Step 2. Make sure the configuration files in the config folder have been properly set up, and confirm that the singularity command can be run successfully.

Step 3. Before officially running the workflow, perform a dry run to check the integrity of the process.

```shell
singularity exec -B ${PWD}:/workDir -B /share:/share \
-e /share/home/salus/salusts.v1.0.sif \
bash -c "export PATH=/opt/conda/bin/:$PATH && \
export LD_LIBRARY_PATH=/opt/conda/lib/:$LD_LIBRARY_PATH && \
cp -rf /app/workflow /workDir/ && cd /workDir/ && \
snakemake --dryrun --printshellcmds"
```

Step 4. Run the workflow.

If running on a single server, you can execute the following command:

```shell
singularity exec -B ${PWD}:/workDir -B /share:/share \
-e /share/home/salus/salusts.v1.0.sif \
bash -c "export PATH=/opt/conda/bin/:$PATH && \
export LD_LIBRARY_PATH=/opt/conda/lib/:$LD_LIBRARY_PATH && \
cp -rf /app/workflow /workDir/ && cd /workDir/ && \
snakemake --cores 16 2>> run.log&"
```

Running in the cluster

If you are submitting tasks on a cluster where the job management system is Slurm, you can use the sbatch command to submit job-name.sh and use the squeue command to check all tasks. Task submission can be performed on the login node mn01. Enter the working directory for this analysis task and save the following code in the job-name.sh text file.

salusSTS.sh Shell

```
#!/bin/bash
#SBATCH --partition=512mem
#SBATCH -n 16           # The number of CPUs used should be consistent with the --cores parameter.
#SBATCH -N 1            # Task priority
#SBATCH -w c001         # Specify the nodes where tasks are executed in the cluster
#SBATCH -J 8Sp0226      # The name of the task can be customized.

export PATH="$PATH:/share/src/third_party/singularity/singularity-ce-3.10.4/bin"

singularity exec -B ${PWD}:/workDir -B /share:/share \
-e /share/home/salus/salusts.v1.0.sif \
bash -c "export PATH=/opt/conda/bin/:$PATH && \
export LD_LIBRARY_PATH=/opt/conda/lib/:$LD_LIBRARY_PATH && \
cp -rf /app/workflow /workDir/ && cd /workDir/ && \
snakemake --cores 16 >> run.log 2>&1"
```

Resubmit a failed process

SalusSTS supports breakpoint resumption. If the workflow is interrupted for any reason, incomplete intermediate files from the current step will be deleted.
Before re-executing the workflow command, you need to run the **snakemake --unlock** command to unlock file protection. After submitting the task again, the workflow will continue from the last completed step.

Manual Alignment HE

Step 1. Generate preprocessed images for manual alignment in ImageJ

When the HEtifPath in the config.yaml file points to the location where the original HE images are stored and TissueSegmentHE is set to True, the workflow will automatically read the color HE images and scale them to grayscale images with the same resolution as the binned chip gene expression images. Users can also generate these images manually:

HE_read.py Python

```
singularity exec -B ${PWD}:/workDir -B /share:/share \
-e /share/home/salus/salusts.v1.0.sif \
bash -c "export PATH=/opt/conda/bin/:$PATH && \
export LD_LIBRARY_PATH=/opt/conda/lib/:$LD_LIBRARY_PATH && \
cd /workDir/ && \
python /workflow/scripts/HE_read.py \
-p /results/04bingrids/{sampleID}_{chipZone}_b{binsize}/ \
-i /HEtifPath/Rat1.tif \
-o /results/ManualHEmask/pre_{sampleID}_{chipZone} \
-b 100 \
-f 0.1
```

Parameters	Description
-p	Specifies the path to the bin-segmented sparse matrix file, which must be consistent with the -b parameter.
-i	HE image at original resolution
-o	Output file directory; the current resolution will be used as the standard reference for contours
-b	Specifies the bin resolution used for segmentation
-f	Specifies the scaling factor for the HE image; a value less than 1 will reduce the image size.

The screen will display the recommended image contour segmentation threshold value from the OTSU algorithm, such as "Recommended threshold_value:109."

This threshold is used in HE_mask.py to binarize the registered H&E grayscale image, and users can modify the threshold value if needed. The workflow will generate mask1-5 to show the results using different denoising and contour closure parameters, making it easier to select the appropriate parameters.

Python

```
/results/ManualHEmask/pre_{sampleID}_{chipZone}/
├── adata_original.h5ad
├── GeExmap.tif             # Gene expression heatmap, as the target for im
agej registration
├── HEimg_sub.tif           # Zoomed out image of HE
├── mask1.png               # User selects the satisfactory parameters acco
rding to the preprocessing results mask1-5
├── mask2.png
├── mask3.png
├── mask4.png
└── mask5.png
```

Step 2.1. Manual registration in ImageJ (choose any method)

Recommended online tutorial (https://www.ai2news.com/blog/3089995/)

Fiji ImageJ → Plugins → BigDataViewer → BigWarp

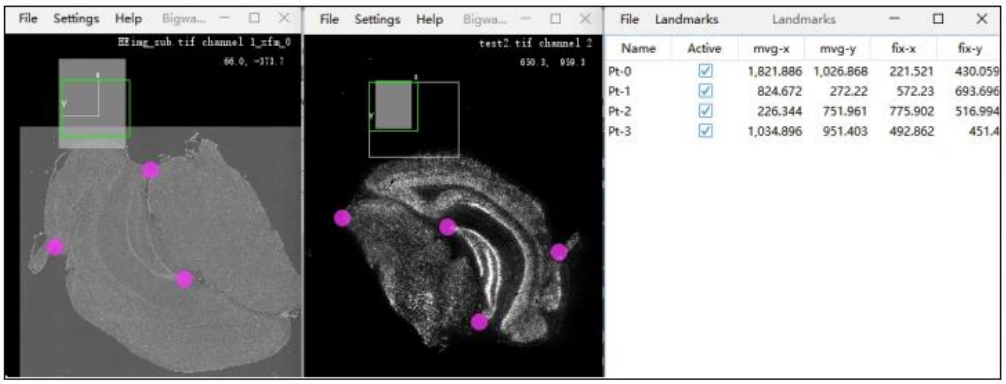
ImageJ download link (https://imagej.net/software/fiji/downloads)

Select the H&E image as the moving image and the sequencing result heatmap as the target image. The goal is to ensure the resolution of the registered image matches that of the sequencing result heatmap.

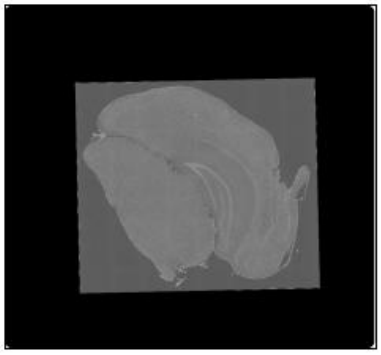
If the sequencing result heatmap appears dim, you can adjust its brightness using the toolbar: Image → Adjust → Brightness/Contrast.

Press the spacebar to toggle marking mode on/off. Use the right mouse button to move the image and the left mouse button to add landmark points. You must select four landmark registration points and export the landmarks.csv file and warped image from the registration page on the right side of the following image.

Note that the resolution of the registered image should be consistent with GeExmap.tif. After the warped image is generated in the viewer, you need to click on the image and then save it from the main ImageJ toolbar.



(Registered image)



Sources and display modes

The viewer can display files containing multiple sources.

The viewer has two display modes, *single-source* and *fused*. In *single-source* mode, only the *current* source is shown (there is exactly one current source at any given time.) In *fused* mode, all *active* sources are shown overlaid.

P	Open source card for editing source visualization modes, color, brightness, and contrast.
Shift P or **Escape**	Close source card.
F	Switch between *single-source* and *fused* mode.
G	Toggle source *grouping*.
I	Switch between tri-linear and nearest-neighbor interpolation.
T	Toggle whether moving image is displayed warped or raw.
Ctrl Shift T	Print a representation of the transformation.
S	Show Brightness & Color dialog.
F2	Show transform type selection dialog.
F3	Show moving image panel Visibility & and Grouping dialog.
F4	Show target image panel Visibility & and Grouping dialog.
U	Show warp visualization grid dialog.
Ctrl O	Open landmarks from a file
Ctrl S	Save landmarks to a file
Ctrl Q	Quick save landmarks
Ctrl E	Export the transformed moving image.
Ctrl Shift E	Export a BDV/XML of the transformed moving image.
Ctrl W	Export displacement (warp) field as an ImagePlus.
Ctrl A	Print affine transformation (or affine part of warp transformation) to the log / terminal.

Step 2.2. Overlaying Layers in Photoshop

Step 2.2.1: Open and Convert to Grayscale

1. Launch Photoshop and open your image by using **File > Open** to select the color image you want to convert to grayscale.

2. Convert to grayscale: Choose **Image > Adjustments > Desaturate**, or use the shortcut **Shift + Ctrl + U** (Windows) or **Shift + Cmd + U** (Mac). Alternatively, select **Image > Mode > Grayscale** and click **Discard** to remove the color information.

3. Add an adjustment layer: At the bottom of the Layers panel, click the **Create new fill or adjustment layer** icon and choose **Black & White**.

4. Adjust parameters: In the Properties panel, you can adjust the sliders for each color channel to achieve the best grayscale effect.

Step 2.2.2: Invert Black and White Colors

1. Select the layer: In the **Layers** panel, make sure the layer you have already converted to grayscale is selected.

2. Invert the colors: Choose **Image > Adjustments > Invert**, or use the shortcut **Ctrl + I** (Windows) or **Cmd + I** (Mac).

Step 3: Save the inverted grayscale image

Save the image: Use **File > Save As...** or **File > Export > Export As...** to choose the save location and file name (for example, **inverted_gray_image.tif**). Upload the registered HE images for each sample to the folder **/results/ManualHEmask/pre_{sampleID}_{chipZone}** generated by the workflow.

By following these steps, you can convert a color image to grayscale in Photoshop, invert the black and white colors, and save the final result.

Step 3. Test contour parameters

This step allows users to experiment with different parameters for HE image contour segmentation. Based on the mask1-5 files generated in Step 1, users can select the desired segmentation parameters and generate the mask.tif file for subsequent standardization of contours across different resolutions. The -t parameter specifies the threshold for image binarization, and -m determines the methods for handling contour noise, holes, and edges. The mask.tif file generated in this step will be used to standardize contour shapes at various resolutions. Please note that, in the registered reg.tif image, the tissue region appears as bright white, while the background is black.

```
singularity exec -B ${PWD}:/workDir -B /share:/share \
-e /share/home/salus/salusts.v1.0.sif \
bash -c "export PATH=/opt/conda/bin/:$PATH && \
export LD_LIBRARY_PATH=/opt/conda/lib/:$LD_LIBRARY_PATH && \
cd /workDir/ && \
python /workflow/scripts/HE_mask.py \
-p /results/04bingrids/{sampleID}_{chipZone}_b{binsize}/ \
-i /results/ManualHEmask/pre_{sampleID}_{chipZone}/reg.tif \
-o /results/ManualHEmask/pre_{sampleID}_{chipZone} \
-b 100 \
-t 109 \
-m 5
```

Parameters	Description
-p	Specifies the path to the bin-segmented sparse matrix file, which must be consistent with the -b parameter.
-i	HE image uploaded by the user after registration

-o	Output file directory; the current resolution will be used as the standard reference for contours

-b	Specifies the bin resolution used for segmentation
-t	The threshold value for image binarization segmentation; it is recommended to use the threshold suggested in Step 1.
-m	Preprocess image contours using different parameter sets, including combinations of dilation and erosion.

```
/results/ManualHEmask/pre_{sampleID}_{chipZone}/
├── adata_in_tissue.h5ad    # The object after contour segmentation
├── adata_original.h5ad     # The object of entire chip zone area
├── combined_image.png
├── figures
│   ├── scatter_n_genes_by_counts.png
│   ├── scatter_pct_counts_mt.png
│   ├── violin_n_genes_by_counts.png
│   ├── violin_pct_counts_mt.png
│   ├── violin_pct_counts_rb.png
│   └── violin_total_counts.png
├── GeExmap.tif
├── HEimg_sub.tif
├── mask1.png
├── mask2.png
├── mask3.png
├── mask4.png
├── mask5.png
├── mask.tif                # The contour shape array resolution is bin 1, 255 for edges and 0 for tissue
├── n_genes_by_counts.png   # Tissue heatmap  after contour segmentation
├── seg.png                 # Tissue heatmap with segmented contours
├── stat.txt
└── total_counts.png        # Tissue heatmap after contour segmentation
```

Step 4. Use the same contour at different resolutions

When the mask.tif file is generated under the /results/ManualHEmask/pre_{sampleID}_{chipZone} folder and the **ReadManualHEmask** parameter in the config.yaml file is set to True, rerunning the workflow will produce tissue segmentation results and corresponding reports at different resolutions.

Please note: If there are multiple samples in a results folder, you must prepare mask.tif results for each sample to ensure that subsequent steps run correctly; otherwise, errors may occur.

Manually select ROI

Step 1. After importing the registered H&E image into ImageJ, select the region of interest (ROI). In the toolbar, click "Analyze" > "Tools" > "Save XY coordinates" to save your selection as the manualroi.csv file. Make sure the resolution of the imported registered image matches the bin size of the sequencing data.

Step 2. Create a new ROI folder under /workDir on the server, and upload the manualroi.csv file to this folder.

Step 3. Run the barcin.py script, which will output a file containing the original barcode sequences and coordinates corresponding to the binned barcodes.

```shell
singularity exec -B ${PWD}:/workDir -B /share:/share \
-e /share/home/salus/salusts.v1.0.sif \
bash -c "export PATH=/opt/conda/bin/:$PATH && \
export LD_LIBRARY_PATH=/opt/conda/lib/:$LD_LIBRARY_PATH && \
cd /workDir/ && \
python python workflow/scripts/barcin.py \
-i /results/04bingrids/Rat_D_b100/ \
-p /ROI/ \
-o /ROI/ \
-b 100                    # Resolution consistent with the registered im
age was set
```

4. Outputs

Overview

Each successfully completed workflow will generate the following file tree. If the workflow is interrupted, you can check run.log to identify the step and reason for the interruption.

Shell

```
/outputName/
├── config
│   ├── config.yaml
│   └── fqlist.csv
├── logs                    # Store the log files for each step
│   ├── 02filterfq
│   ├── 03mapping
│   ├── 04bingrids
│   ├── 05tissuecut
│   └── 06reports
├── results
│   └── 01fstCoord          # Extracted barcode coordinate files by chipzone
│
│   ├── 02filterfq
│   │   ├── rawFq
│   │   ├── cleanFq         # Reads after quality control
│   │   ├── cutFq           # Reads after cut adpter
│   │   ├── correct         # If set BCUMI2pass true will generate this fil
e, allow 3 base mismatch
│   │   └── merge           # R1 and R2 after quality control, cut adpter an
d barcode correction, will be import to STARsolo
│   ├── 04bingrids          # cellbin and squarebin sparse expression matric
es with different resolutions were stored, along with manual HE segmentati
on mask.tif
│   │   ├── Rat1_D2_b40
│   │   ├── Rat1_D2_b100
│   │   ├── Rat1_D2_b400
│   │   └── Rat1_D2_cellbin
│   ├── 05tissuecut         # Results of tissue segmentation based on sequen
cing results or HE
│   │   ├── Rat1_D2_b40
│   │   ├── Rat1_D2_b100
│   │   ├── Rat1_D2_b400
│   │   └── Rat1_D2_cellbin
│   └── 06reports           # Store the result report file
│       ├── qc_stats
│       ├── saturation
│       └── html            # Report files summarizing results at different
resolutions
├── workflow                # Store rules of snakemake
└── run.log                 # Record the program running log to facilitate e
rror elimination
```

Results

The detailed results file tree is as follows:

Shell

```
results
├── 01fstCoord
│   ├── fstBC_D2.zoned.spatial.txt.gz       # The file contains three columns, barcode, X coordinate, Y coordinate
│   └── fstBC_D2.zoned.whitelist.txt.gz     # barcode sequence at the D2 chip zone
├── 02filterfq
│   ├── cleanFq
│   │   ├── newCoord_fstBC_X111_DualLower.fastp.fq.gz
│   │   ├── snd_Rat1_D2_f000.R1.fq.gz       # Reads after QC, first 30 bases are barcode and last 10 bases are UMI
│   │   └── snd_Rat1_D2_f000.R2.fq.gz       # Reads after QC, contain cDNA sequences
│   ├── cutFq
│   │   ├── sndBC_Rat1_D2_f000.cutadapt.fq.gz   # Paired read with sndBC barcode and UMI
│   │   └── sndNA_Rat1_D2_f000.cutadapt.fq.gz   # Reads after QC,  adapter trimmed cDNA sequence
│   ├── correct
│   │   └── corrected_Rat1_D2_f000          # Paired read with sndBC barcode and UMI
│   │      └──sndBC.stat.txt                # Summary count of sndBC corrected reads
│   └── merge
│       ├── sndBC_Rat1_D2.merged.fq.gz      # Paired reads with sndBC barcode and UMI, used in STAR alignment
│       └── sndNA_Rat1_D2.merged.fq.gz      # Paired reads with cDNA sequence, used in STAR alignment
├── 03mapping
│   ├── Rat1_D2           # If fqlist.csv contains multiple samples, multiple folders are generated, each named [sampleID]_[chipzone]
│   │   ├── Aligned.sortedByCoord.out.bam
│   │   ├── Log.final.out # log of the alignment
│   │   ├── Log.out       # Log of STAR running time and memory consumption
│   │   ├── Log.progress.out                # log of STAR running process
│   │   └── Solo.out
│   │       ├── Barcodes.stats              # Log of R1 alignment was recorded by STAR
│   │       ├── Gene      # Simply considers overlap of an alignment with the exons
│   │       │   ├── Features.stats          # Log of R2 alignment was recorded by STAR
```

```
|   |       |   ├── raw   # File structure is consistent with the single-
cell sparse matrix structure of 10X genomics
|   |       |   |   ├── barcodes.tsv.gz
|   |       |   |   ├── features.tsv.gz
|   |       |   |   └── matrix.mtx.gz       # Statistics of the alignment
|   |       |   └── Summary.csv
|   |       ├── GeneFull_Ex50pAS            # This options is most simila
r to CellRanger's intronic count option
|   |       |   ├── Features.stats
|   |       |   ├── raw
|   |       |   |   ├── barcodes.tsv.gz
|   |       |   |   ├── features.tsv.gz
|   |       |   |   ├── matrix.mtx.gz
|   |       |   └── Summary.csv
|   |       ├── spatial.txt
|   |       └── Velocyto                    # The RNA cleavage informatio
n was recorded in the folder
|   |           ├── Features.stats
|   |           ├── raw
|   |           |   ├── ambiguous.mtx.gz
|   |           |   ├── barcodes.tsv.gz
|   |           |   ├── features.tsv.gz
|   |           |   ├── spliced.mtx.gz
|   |           |   └── unspliced.mtx.gz
|   |           └── Summary.csv
|   └── Rat1_D2_pre
├── 04bingrids                             # The sparse expression matri
x after square bin is recorded in the folder
|   ├── Rat1_D2_b100
|   |   ├── barcodes.tsv.gz
|   |   ├── bc.tsv.gz                      # barcode correspondence betw
een the original barcode and the square bin barcode
|   |   ├── features.tsv.gz
|   |   ├── matrix.mtx.gz
|   |   └── spatial.txt.gz                 # The coordinates of the barc
ode were recorded in the same order with barcoedes.tsv.gz
|   ├── Rat1_D2_b40
|   |   ├── barcodes.tsv.gz
|   |   ├── bc.tsv.gz
|   |   ├── features.tsv.gz
|   |   ├── matrix.mtx.gz
|   |   └── spatial.txt.gz
├── 05tissuecut                            # Store the results after aut
omatic tissue segmentation
|   ├── Rat1_D2_b100
|   |   ├── adata_in_tissue.gem.gz         # The gem file for Stereo-se
q, containing the information after binning
```

```
│   │   ├── adata_in_tissue.h5ad            # The h5ad format file for sc
anpy corresponds to the tissue coverage area
│   │   ├── adata_original.h5ad             # The h5ad format file for sc
anpy corresponds to all regions
│   │   ├── combined_image.png              # Summary of quality control
chart
│   │   ├── figures
│   │   │   ├── color_bar.png
│   │   │   ├── left.png
│   │   │   ├── right.png
│   │   │   ├── scatter_n_genes_by_counts.png
│   │   │   ├── scatter_pct_counts_mt.png
│   │   │   ├── violin_n_genes_by_counts.png
│   │   │   ├── violin_pct_counts_mt.png
│   │   │   ├── violin_pct_counts_rb.png
│   │   │   └── violin_total_counts.png
│   │   ├── img.png
│   │   ├── index.txt
│   │   ├── in_tissue.png
│   │   ├── n_genes_by_counts_color_bar.png
│   │   ├── n_genes_by_counts.png
│   │   ├── seg.png
│   │   ├── stats.txt
│   │   ├── total_count_color_bar.png
│   │   └── total_counts.png
│   ├── Rat1_D2_b40
│   │   ├── adata_in_tissue.gem.gz
│   │   ├── adata_in_tissue.h5ad
│   │   ├── adata_original.h5ad
│   │   ├── combined_image.png
│   │   ├── figures
│   │   │   ├── left.png
│   │   │   ├── right.png
│   │   │   ├── scatter_n_genes_by_counts.png
│   │   │   ├── scatter_pct_counts_mt.png
│   │   │   ├── violin_n_genes_by_counts.png
│   │   │   ├── violin_pct_counts_mt.png
│   │   │   ├── violin_pct_counts_rb.png
│   │   │   └── violin_total_counts.png
│   │   ├── img.png
│   │   ├── index.txt
│   │   ├── in_tissue.png
│   │   ├── n_genes_by_counts_color_bar.png
│   │   ├── n_genes_by_counts.png
│   │   ├── seg.png
│   │   ├── stats.txt
│   │   ├── total_count_color_bar.png
```

```
114 │   │   └── total_counts.png
115 │   ├── Rat1_D2_stats.txt
116 │   ├── Rat1_D2_cellbin
117 │   │   ├── adata_in_tissue.h5ad
118 │   │   ├── barcodes.tsv.gz
119 │   │   ├── bc.tsv.gz
120 │   │   ├── cellRNA.extended.png
121 │   │   ├── cellRNA.original.png
122 │   │   ├── combined_image.png
123 │   │   ├── config.sh
124 │   │   ├── config.txt
125 │   │   ├── density.txt
126 │   │   ├── expr.b50.x0.y0.w816.h812.png
127 │   │   ├── expr.b5.x1940.y2160.w4410.h3850.mat.txt
128 │   │   ├── expr.b5.x1940.y2160.w4410.h3850.png
129 │   │   ├── features.tsv.gz
130 │   │   ├── figures
131 │   │   │   ├── color_bar.png
132 │   │   │   ├── left.png
133 │   │   │   ├── part1.png
134 │   │   │   ├── part2.png
135 │   │   │   ├── part3.png
136 │   │   │   ├── part4.png
137 │   │   │   ├── part5.png
138 │   │   │   ├── right.png
139 │   │   │   ├── scaled.png
140 │   │   │   ├── scatter_n_genes_by_counts.png
141 │   │   │   ├── scatter_pct_counts_mt.png
142 │   │   │   ├── violin_n_genes_by_counts.png
143 │   │   │   ├── violin_pct_counts_mt.png
144 │   │   │   ├── violin_pct_counts_rb.png
145 │   │   │   └── violin_total_counts.png
146 ├── 06reports
147 │   ├── html
    │   │   └── merged_Rat1_D2
148 │   │       └── report.html                # Different resolutions' resu
149 lts are stored in one report file
150 │   └── saturation
    │       └── Rat1_D2
151 │           └── saturation.tsv             # Saturation curve informatio
```

Logs

Shell

```
logs
├── 02filterfq
│   ├── cleanFq
│   │   ├── fstBC_X111_DualLower.fastp.log
│   │   ├── fstBC_X111_DualUpper.fastp.log
│   │   ├── snd_Rat1_D2.fastp.log
│   │   └── snd_Rat1_D2.html
│   ├── cutFq
│   │   └── sndNA_Rat1_D2.cutadapt.json
│   └── fst
│       ├── fstBC_X111_DualLower.html
│       ├── fstBC_X111_DualLower.json
│       ├── fstBC_X111_DualUpper.html
│       └── fstBC_X111_DualUpper.json
├── 03mapping
│   └── Rat1_D2.log
├── 05tissuecut
│   ├── Rat1_D2b40.log
│   ├── Rat1_D2b100.log
│   └── Rat1_D2_cellbin.log
├── 06reports
│   └── saturation
│       └── Rat1_D2.log
├── pyqc.sh
├── segmentation
│   └── pre_Rat1_D2.log
└── star
    └── Rat1_D2.sort.log
```