Mistic

Sandhya Prabhakaran

Jan 28, 2023

CONTENTS:

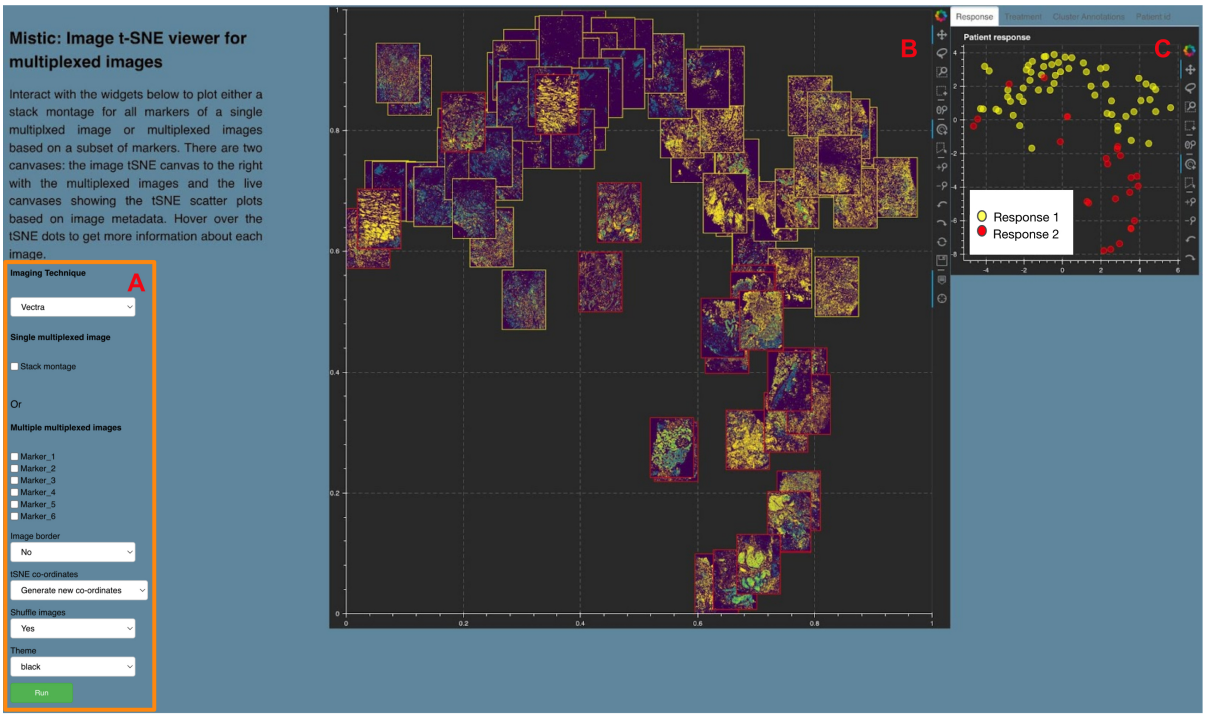
CHAPTER

ONE

INTRODUCTION

1.1 What is Mistic

This is a Python tool using the Bokeh library to view multiple multiplex images simultaneously. The code has been tested on 7-panel Vectra TIFF, 32- & 64-panel CODEX TIFF, 16-panel CODEX QPTIFF, 4-panel CyCIF TIFF, and 44-panel t-CyCIF TIFF images.



A sample Mistic GUI with user inputs is shown. **A.** User-input panel where imaging technique choice, stack montage option or markers can be selected, images borders can be added, new or pre-defined image display coordinates can be chosen, and a theme for the canvases can be selected. **B.** Static canvas showing the image t-SNE colored and arranged as per user inputs. **C.** Live canvas showing the corresponding t-SNE scatter plot where each image is represented as a dot. The live canvas has tabs for displaying additional information per image. Metadata for each image can be obtained by hovering over each dot.

1.2 Motivation behind developing Mistic

- Multiplex imaging of tissues, allows the simultaneous imaging of multiple biomarkers on a tissue specimen of interest, and is a critical tool for clinical cancer diagnosis and prognosis.
- There are many commercial and open-source software to view and process single images where processing could involve normalization, segmentation, tiling, and image processing to extract and prepare single cell images for actual analysis. Many of the images are large data files.
- The value in simultaneously viewing such multiple images is to get a preliminary overview of existing patterns in the data such as presence of multiple spatially- distributed cell phenotypes, and the tissue architecture.
- This exploratory understanding will also enable viewing a specific marker expression pattern across all images, helps to identify images expressing a particular phenotype or to select images for subsequent downstream analysis.
- Mistic aims to fill this gap as there are currently no freely available tools providing this functionality.

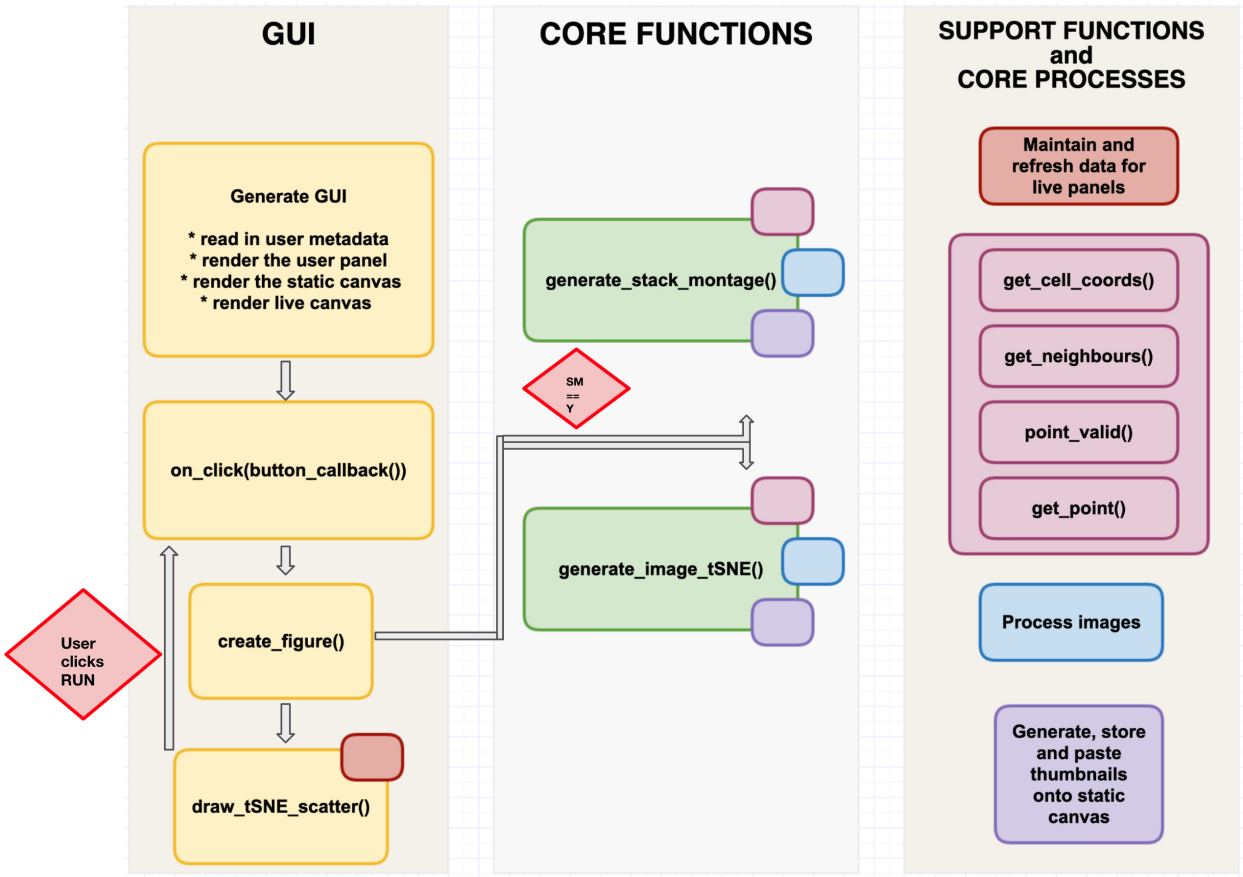
1.3 Mistic features

- Two canvases:
  - static canvas with the image tSNE rendering
  - live canvases with tSNE scatter plots for image metadata rendering
- Dropdown option to select the imaging technique: Vectra, CyCIF, t-CyCIF, or CODEX
- Option to choose between Stack montage view or multiple multiplexed images by selecting the markers to be visualised at once
- Option to place a border around each image based on image metadata
- Option to use a pre-defined tSNE or generate a new set of tSNE co-ordinates
- Option to shuffle images with the tSNE co-ordinates
- Hover functionality available on the tSNE scatter plot to get more information of each image
- Save, zoom, etc each of the Bokeh canvases

CHAPTER

TWO

FUNCTIONS



Flowchart sketching flow between functions and processes in Mistic. (SM = Stack montage, Y = Yes, N = No)

2.1 get_cell_coords(pt,a)

Get the coordinates of the cell that pt = (x,y) falls in.

```
def get_cell_coords(pt,a):
        return int(pt[0] // a), int(pt[1] // a)
```

Parameters:

`pt` : `float` 2D coordinates of a point

`a :` `float` scaled minimum distance between cells

Output

`list` coordinates of the cell that pt lies in

2.2 get_neighbours(coords,nx,ny,cells)

Return the indexes of points in cells neighbouring cell at coords. For the cell at coords = (x,y), return the indexes of points in the cells with neighbouring coordinates illustrated below: ie those cells that could contain points closer than r

```
def get_neighbours(coords,nx,ny,cells):
    # Refer main.py on Github
     return neighbours
```

Parameters:

`coords :` `float` 2D coordinates of a point

`nx, ny :` `int` Number of cells in the x- and y-directions of the grid

`cells : dictionary`
Dictionary of cells or grid squares: each key is a cell's coordinates, the corresponding value is the index of that cell's coordinates in the samples list (or None if the cell is empty)

Output

`neighbours:` `list` List of neighbouring cells for coords

2.3 point_valid(pt,a,nx,ny,cells,samples,r)

Returns True or False based on if the pt is a valid point to be added to samples. It must be no closer than r from any other point by checking the cells in its immediate neighbourhood.

```
def point_valid(pt,a,nx,ny,cells,samples,r):
    # Refer main.py on Github
     return True (or False)
```

Parameters:

`pt :` `float` 2D coordinates of a point

`a :` `float` scaled minimum distance between cells

`nx, ny :` `int` Number of cells in the x- and y-directions of the grid

`cells : dictionary`
Dictionary of cells or grid squares: each key is a cell's coordinates, the corresponding value is the index of that cell's coordinates in the samples list (or None if the cell is empty)

`samples : list`
List with valid neighbouring coordinates for a given point pt

`r :` `int` Minimum distance between samples. Set to 2

Output

`boolean` Returns True or False if sampled point is eligible to be pt's neighbour

2.4 get_point(k, refpt,r,a,nx,ny,cells,samples)

Try to find a candidate point relative to refpt to emit in the sample. We draw up to k points from the annulus of inner radius r, outer radius 2r around the reference point, refpt. If none of them are suitable (because they're too close to existing points in the sample), return False. Otherwise, return the pt.

```
def get_point(k, refpt,r,a,nx,ny,cells,samples):
     # Refer main.py on Github
     return True (or False)
```

Parameters:

`k :` `int` number of candidate points sampled arounf the reference point, refpt

`refpt :` `float` 2D coordinates of the reference point

`a :` `float` scaled minimum distance between cells

`nx, ny :` `int` Number of cells in the x- and y-directions of the grid

`cells : dictionary`
Dictionary of cells or grid squares: each key is a cell's coordinates, the corresponding value is the index of that cell's coordinates in the samples list (or None if the cell is empty)

`samples : list`
List with valid neighbouring coordinates for a given point pt

`r :` `int` Minimum distance between samples. Set to 2

Output

`boolean` Returns True or False if candidate point is eligible to be refpt's neighbour

---

Note: The above functions: `get_cell_coords()`, `get_neighbours()`, `point_valid()`, `get_point()` are modified from https://scipython.com/blog/poisson-disc-sampling-in-python/

2.5 button_callback()

Function that is called when user clicks 'Run' from the GUI

```
def button_callback():
     # Refer main.py on Github
     return ([p,p1,p2,p3,p4, source, tabs])
```

Output

`p :` `figure` Static figure canvas with the image tSNE

`p1, p2, p3, p4 :` `figure` Live canvas with tSNE scatter plots for each of the image metadata

`source :` `ColumnDataSource, dictionary` Dictionary of metadata and associated variables (like colour, thumbnail location) for each image which is maintained and updated based on user preferences for each 'Run' episode

`tabs :` `Tabs` Tabs of Panels where each panel corresponds to an image metadata

2.6 create_figure(stack_montage_flag)

Function collects the user choices from the GUI and calls either the generate_stack_montage() for reading in a single image or the generate_image_tSNE() for multiplexed images

```
def create_figure(stack_montage_flag):
     # Refer main.py on Github
     return ([p,tsne_points, file_name_hover,markers_single, cluster_ms_list])
```

Parameters:

`stack_montage_flag` : `bool` If True, the generate_stack_montage() is called, else the generate_image_tsne() is called.

Output

`p` : `figure` Static figure canvas with the image tSNE

`tsne_points` : `float` 2D coordinates for each image where coordinates can be randomly generated/arranged in rows/user defined. The coordinates are generated based on number of images and size (length and breadth) of the static canvas

`file_name_hover` : `str` file name with path to populate the 'thumbnail' entry in the Hover tool

`markers_single` : `list` list of markers selected by the user

`cluster_ms_list` : `list` list of channel names necessary to populate the hover tool in the live panels

2.7 generate_stack_montage(chk_box_marker_sm, rb_imtech_val, LABELS_MARKERS)

Function generates a stack montage by using each marker channel of a multiplexed image.

- Reads in and pre-processes the image channels
- Generates evenly-spaced points on the static canvas to arrange the images in rows
- Generates thumbnails, and pastes these onto the static canvas
- Stores the thumbnails in the output folder
- Updates the hover tool with thumbnail paths, marker names and metadata

```
def generate_stack_montage(chk_box_marker_sm, rb_imtech_val, LABELS_MARKERS):
     # Refer main.py on Github
     return([file_name_rot,tsne, file_name_hover])
```

Parameters:

`chk_box_marker_sm` : `int` If checkbox is checked by user on the GUI, chk_box_marker_sm = 1 indicating that the stack montage option is selected. If checkbox is unchecked, the multiple image tSNE generation process proceeds based on markers chosen by the user using the GUI

`LABELS_MARKERS` : `list` List of all marker channels in the multiplexed images. This is provided by the user in user_inputs/metadata folder as Markers.csv. User can still choose a subset of LABELS_MARKERS, through the GUI, for visualizing the image tSNE

`rb_imtech_val` : `int` If value = 0, Vectra processing is invoked, if value = 1, t-CyCIF processing is invoked and if value = 2, CODEX processing is invoked

Output

`file_name_rot` : `str` file name with path where the final image with thumbnails is located

`tsne` : `float` 2D coordinates for each image to be rendered in rows. The coordinates are generated based on number of images and size (length and breadth) of the static canvas

`file_name_hover` : `str` file name with path to populate the 'thumbnail' entry in the Hover tool

2.8 generate_image_tSNE(chk_box_marker,rb_val,rb_rs_val, rb_shf_val,rb_imtech_val,mc,wc,LABELS_MARKERS)

Function generates the image tSNE using the multiplexed images and based on user inputs

- Reads in and pre-processes the images
- Generates random or evenly-spaced points on the static canvas to arrange the images in rows/Reads in the user-provided tSNE
- Generates thumbnails, and pastes these onto the static canvas
- Stores the thumbnails in the output folder
- Updates the hover tool with thumbnail paths, marker names and metadata
- shuffle or no shuffle option is handled in this function where images are randomly shuffled

```
def generate_image_tSNE(chk_box_marker,rb_val,rb_rs_val,rb_shf_val,rb_imtech_
↪val,mc,wc LABELS_MARKERS):
    # Refer main.py() on Github
    return([file_name_rot,tsne, file_name_hover])
```

Parameters:

`chk_box_marker` : `int` If checkbox is checked by user on the GUI, chk_box_marker = 1 and the user-selected markers are collected by this variable

`rb_val` : `str` Choice of having a border for each image based on image metadata. If 'No' is chosen, no border is set for images

`rb_rs_val` : `str` Choice to create a tSNE based on user-defined points, random coordinates or stack the images in rows

`rb_shf_val` : `str` Choice to shuffle images ('Yes') or not ('No') while rendering the images on the static canvas

`rb_imtech_val` : `int` If value = 0, Vectra processing is invoked, if value = 1, t-CyCIF processing is invoked and if value = 2, CODEX processing is invoked

`mc` : `list` List of all markers available as shown on GUI

`wc` : `list` List of weights for all markers shown on GUI

`LABELS_MARKERS` : `list` List of all marker channels in the multiplexed images. This is provided by the user in user_inputs/metadata folder as Markers.csv. User can still choose a subset of LABELS_MARKERS, through the GUI, for visualizing the image tSNE

Output

`file_name_rot` : `str` file name with path where the final image with thumbnails is located

`tsne :` `float` 2D coordinates for each image where coordinates can be random/arrnaged in rows/user defined. The coordinates are generated based on number of images and size (length and breadth) of the static canvas

`file_name_hover :` `str` file name with path to populate the 'thumbnail' entry in the Hover tool

2.9 draw_tSNE_scatter(tsne1,file_name_hover,cluster_ms_list)

Function that generates the image tSNE scatter plots for the Live canvases

Parameters

`tsne1 :` `float` 2D coordinates for each image where coordinates can be randomly generated/arranged in rows/user defined. The coordinates are generated based on number of images and size (length and breadth) of the static canvas

`file_name_hover :` `str` file name with path to populate the 'thumbnail' entry in the Hover tool

`cluster_ms_list :` `list` list of channel names necessary to populate the hover tool in the live panels

Output

`p1, p2, p3, p4 :` `figure` Live canvas with tSNE scatter plots for each of the image metadata

`source :` `ColumnDataSource, dictionary` Dictionary of metadata and associated variables (like colour, thumbnail location) for each image which is maintained and updated based on user preferences for each 'Run' episode

CHAPTER

THREE

REQUIREMENTS

3.1 Minimum Hardware specifications

Mistic has been tested on:

- CPU: Intel Core i9
- CPU Speed: 2.4 GHz
- Number of CPUs: 1
- Total Number of Cores: 8
- RAM: 32 GB

3.2 Software requirements

- Python >= 3.6
  - Install Python from here.
  - Open a command prompt (or the Terminal application) to download `pip`. Type:
  - `curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py`
  - `python3 get-pip.py` and wait for the installation to complete
  - Verify the `pip` installation by typing `python3 --version`

CHAPTER

FOUR

CODE INSTALLATION

4.1 Download Mistic

- `pip install mistic`
- To download Mistic's code repository, perform one of the three options:
  - Download the code repository from link.
  - Open a Terminal and at the command line, type:

```
$ git clone https://github.com/MathOnco/Mistic.git
```

  - Code can also be downloaded from Zenodo.

4.2 Preload user data

- In the Mistic folder, navigate to /user_inputs folder to upload input files:
  - `Mistic_code/code/user_inputs/`
  - Use the /figures folder to upload the multiplexed images
    - Example NSCLC Vectra dataset is available from Zenodo_data.
  - Use the /metadata folder to upload the markers.csv and imaging markers of interest as Markers_ids.csv
    - Example files are provided in the subfolders: Vectra, CyCIF, t-CyCIF and CODEX
    - Move the files from the relevant subfolder into the /metadata folder
    - Note: For the Stack Montage option, only the markers.csv file is required.
  - (Optional) Use the /metadata folder to
    - Upload image tSNE co-ordinates as X_imagetSNE.csv
      - If no user-generated tSNE co-ordinates are provided, Mistic will generate a set of t-SNE coordinates to render the images
    - Upload image metadata such as
      - Cluster labels as Cluster_categories.csv
      - If cluster labels are not provided, Mistic will cluster the images using a Bayesian mixture model.
      - Patient_ids as Patient_ids.csv

- Treatments as Treatment_catgories.csv
- Patient response as Response_categories.csv
- If any of these are unavailable, Mistic will use either the randomly-generated or user-provided tSNE points without any color coding i.e. dots are colored in gray.
- Sample metadata files are provided for reference in separate subfolders for each imaging technique (Vectra, CyCIF, t-CyCIF and CODEX) in the /metadata folder
- If using the sample metadata, move the files from the relevant subfolder into the /metadata folder

4.3 Run Mistic

- At the command prompt pointing to /code, type
  - `bash mistic.sh`
  - This runs a bokeh server locally and will automatically open the interactive dashboard in your browser at http://localhost:5098/image_tSNE_GUI
  - Enter the imaging format, montage or multiplexed views and other user options on the GUI and click `Run`.
- **Examples for running Mistic:**
  - t-CYCIF data: *Vignettes on t-CyCIF data*
  - Vectra data: *Vignettes on Vectra data*

CHAPTER

FIVE

MISTIC EXPERIMENTS

5.1 Datasets used to test Mistic

Dataset	Format	Dimensions	Size per image	Number of images	Thumbnail size per TIFF	Image generated by
NSCLC FoVs	7-marker TIFF	7 x 1344 x 1008	10-50MB	92	<100 KB	Perkin Elmer Vectra
Lung Cancer lymph node	OME-TIFF	44 x 10101 x 9666	13GB	70	<22 MB	t-CyCIF
Lung Cancer primary	OME-TIFF	44 x 14447.5x 11100.5	322 MB per channel (21.22GB total file size)	44 slides (1 image)	<10 MB	t-CyCIF
Endometrial cancer (Tissue Microarray)	PNG (generated from 7-marker TIFF)	950 x 1200	250-1MB	210	<300 KB	Perkin Elmer Vectra
Human FFPE Tonsil	OME_TIFF	32 x 6720 X 5040	67.7MB per channel (2.17GB total file size)	32 slides	<3.7MB	CODEX
Human FFPE Breast adenocarcinoma	OME_TIFF	64 x 5040 x 9408	6.07 GB	88	8.2 MB	CODEX
Human FFPE Tonsil	QPTIFF	16 x 2760 x 2070 (4th pyramid)	2.45 GB	105	1.1 MB	CODEX
Human colorectal carcinoma	OME_TIFF	4 x 1344 x 1792	7.4 MB	42	~600 KB	CyCIF

Table listing the different datasets Mistic has been tested on.

5.2 Non-small cell lung cancer

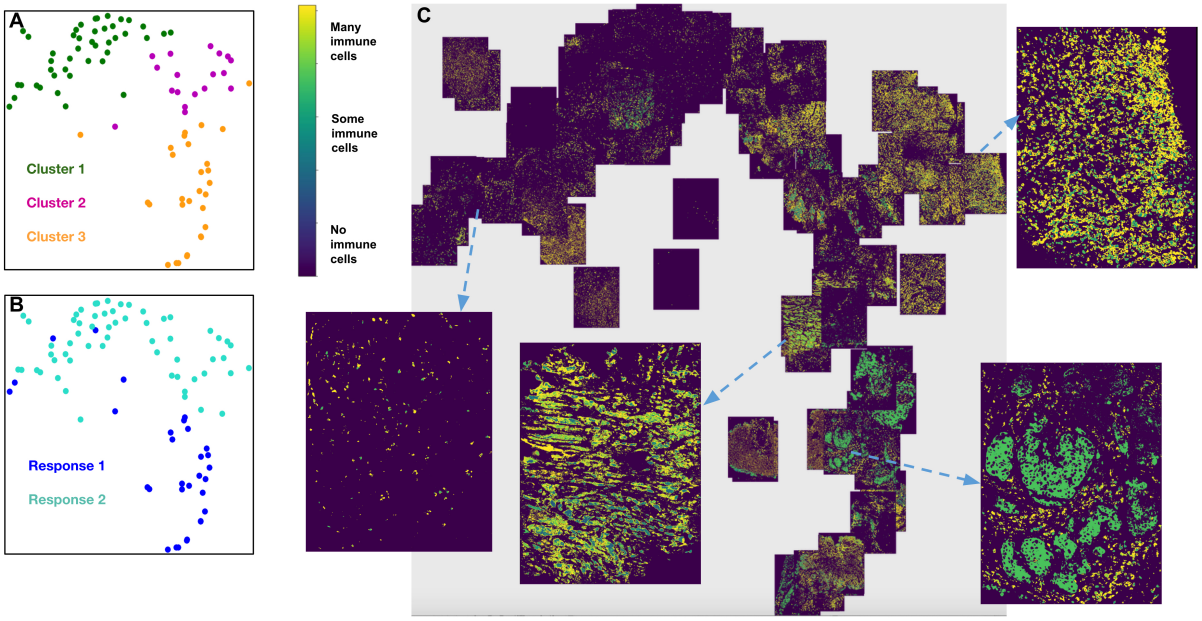


Figure showing Response 1 and Response 2 patients with NSCLC have significantly different cellular compositions. **A.** 2D t-SNE plot showing three clusters annotated with the differentially expressed markers per cluster. Clusters are obtained using the tumor-immune cell counts at the tumor border where the borders are estimated using convex hulls approximations. **B.** Same t-SNE as in (**A**) depicting the disease response spread. **C.** Image t-SNE using same t-SNE coordinates as in (**A**) illustrating the gradient of immune cells across images. A higher colocalization of immune cells (shown in green) is seen for Response 1 patients. Instructions to run Mistic on a toy dataset is here: *Vignettes on Vectra data* 7.1.

5.3 Lung adenocarcinoma metastasis to lymph node

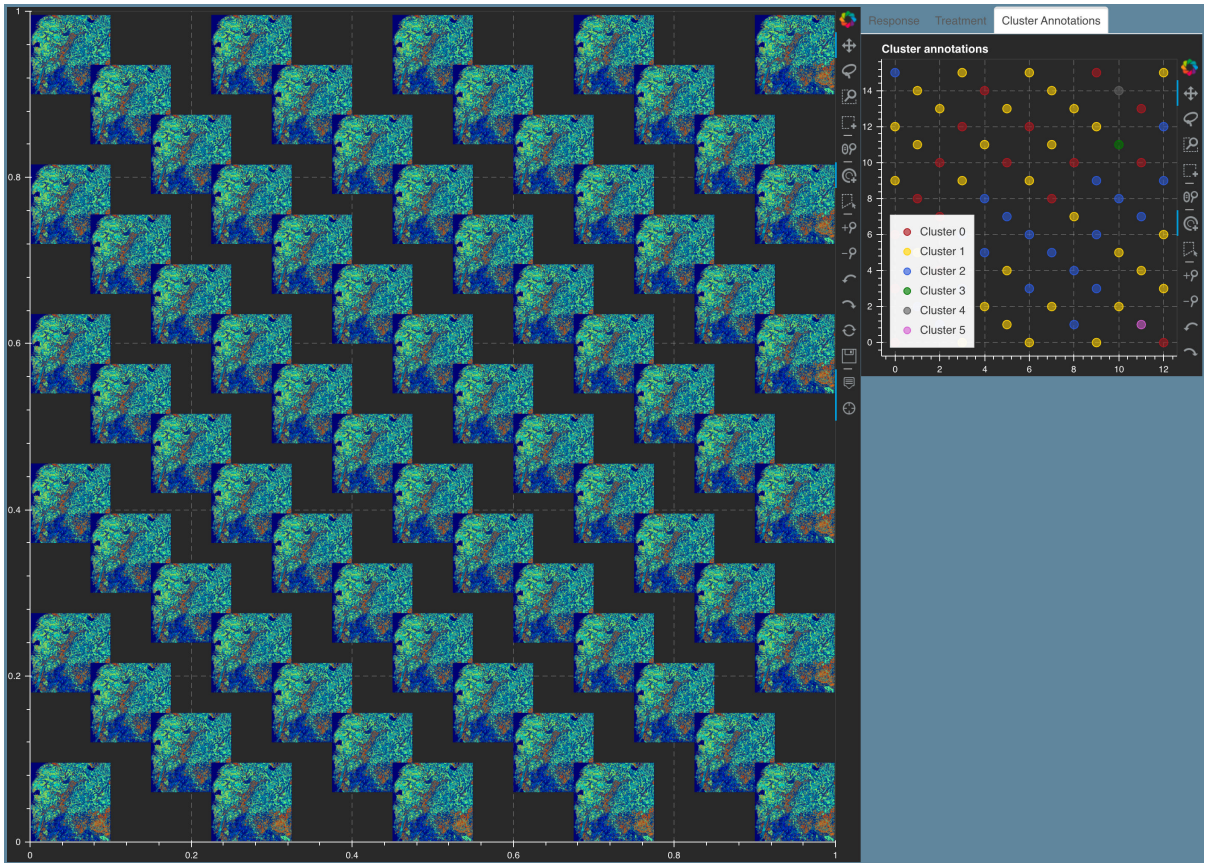


Figure showing Mistic on the Lung adenocarcinoma lymph t-CyCIF data. The static canvas shows 70 repeats of the Lung t-CyCIF image arranged in rows. Mistic gives the composite image using 6 markers (CD45, Keratin, aSMA, FoXP3, PD-1, PD-L1). The live canvas shows proxy cluster assignments. We discuss how to run Mistic on this data here: *Vignettes on t-CyCIF data* 6.1.

5.4 Primary lung squamous cell carcinoma

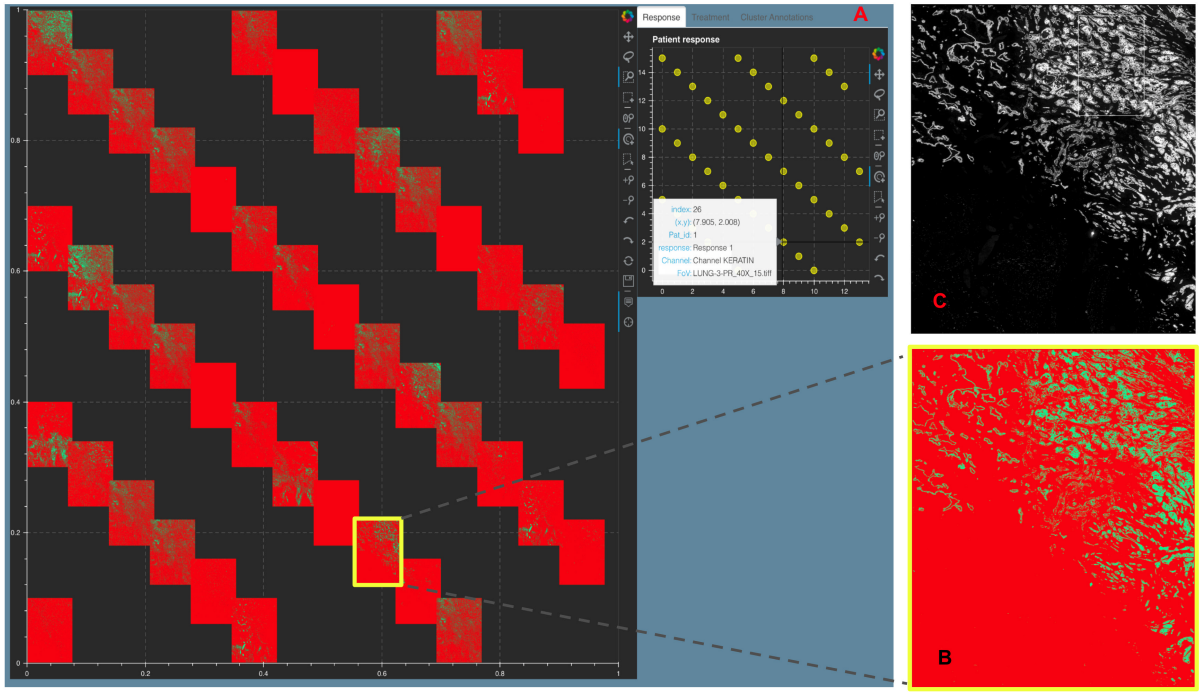


Figure showing a Stack Montage from Mistic for the Primary Lung t-CyCIF data with 44 markers. **A.** The static canvas shows all 44 markers and the live canvas shows the tSNE scatter plot. We identify the Keratin channel using the live canvas (shown with hover tool details) and highlight the Keratin thumbnail in yellow in the static canvas. **B.** The zoomed in Keratin thumbnail (file name obtained from the hover tool) and **C.** The t-CyCIF image for Keratin as viewed using Minerva [45]. Minerva provides the single marker views for 12 markers whereas with Mistic we can view all 44 channels as a montage. We discuss how to run Mistic on this data here: *Vignettes on t-CyCIF data* 6.2.

5.5 Tissue Microarray cores for Endometrial cancer

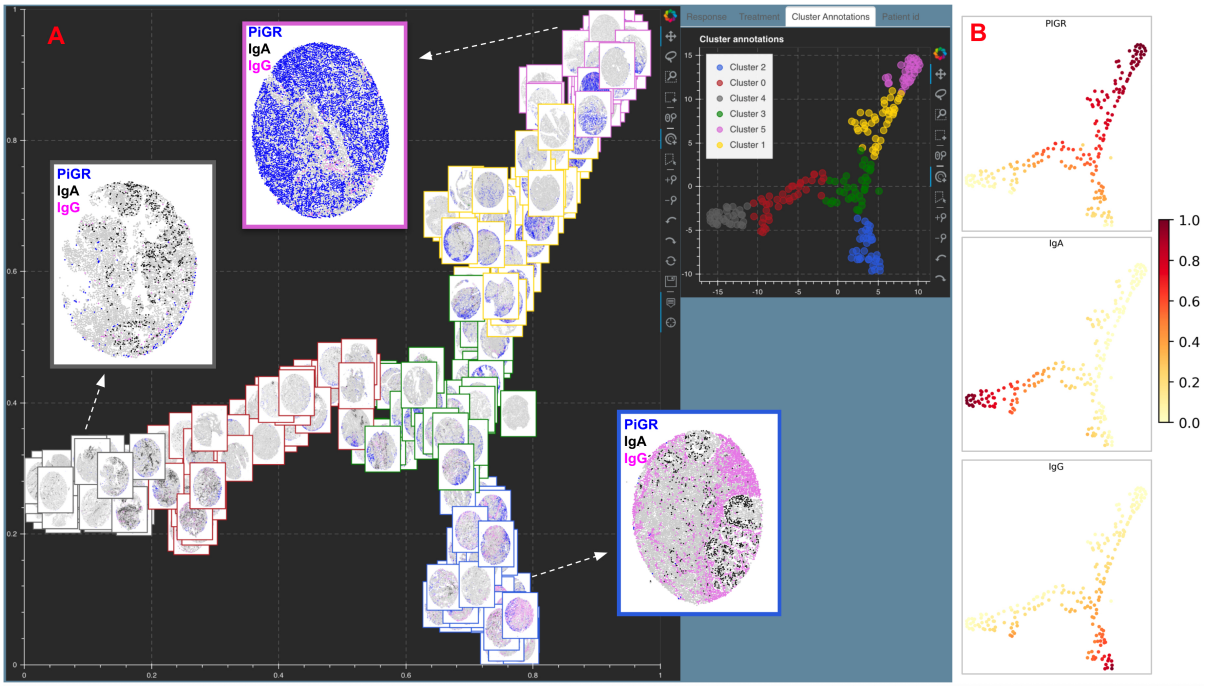


Figure showing **A.** Image tSNE rendering using Mistic for 210 Tissue Microarray (TMA) cores of Endometrial cancer. Each core depicts pIGr on tumor cells (blue), IgA (black), IgG (pink), plasma cells (red), additional B-cells (green) and unstained cells (gray). Each core has a border that matches the cluster it belongs to (see the Cluster annotations live panel). Representative cores from each of the clusters dominated by pIgr, IgA and IgG are shown. **B.** tSNE scatter plot showing the marker spread computed from the 210x5 count matrix prior to clustering.

5.6 Human FFPE Tonsil data

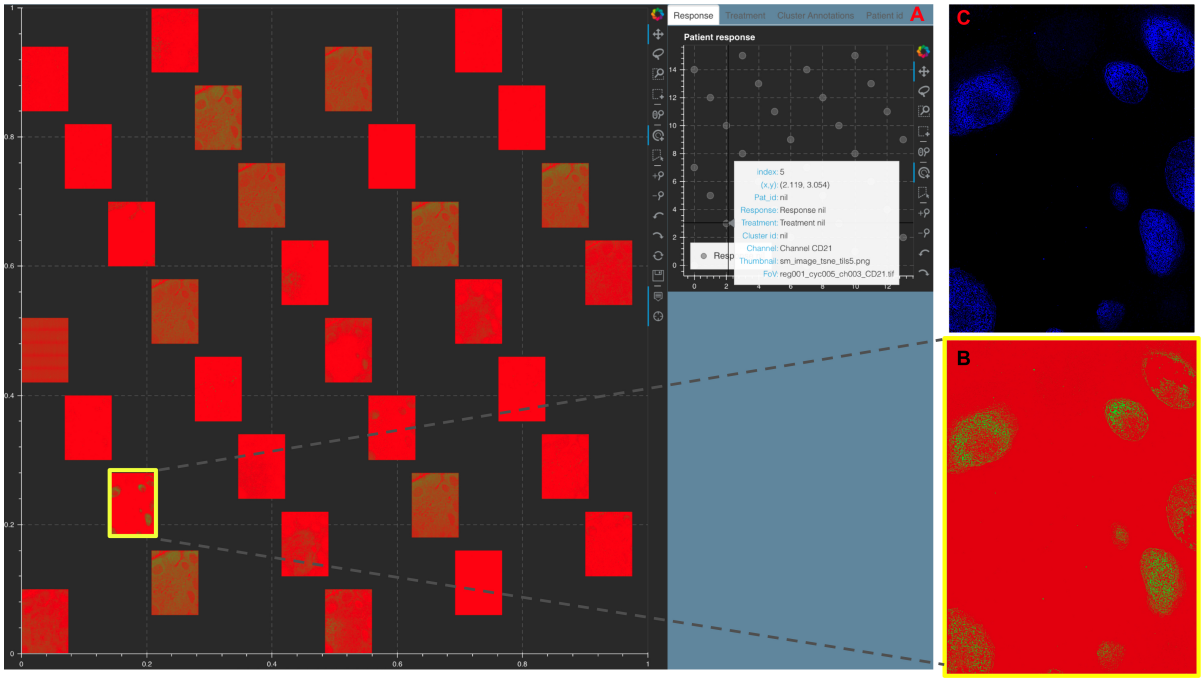


Figure showing a Stack Montage from Mistic for the Human FFPE Tonsil CODEX data with 32 markers. **A.** The static canvas shows all 32 markers and the live canvas shows the tSNE scatter plot. We identify the CD21 channel using the live canvas (shown with hover tool details) and highlight the CD21 thumbnail in yellow in the static canvas. **B.** The zoomed in CD21 thumbnail (file name obtained from the hover tool) and **C.** The CODEX CD21 channel as viewed using FIJI. FIJI provides the single marker views for 32 markers whereas with Mistic we can view all 32 markers as a montage.

5.7 Human FFPE Breast adenocarcinoma

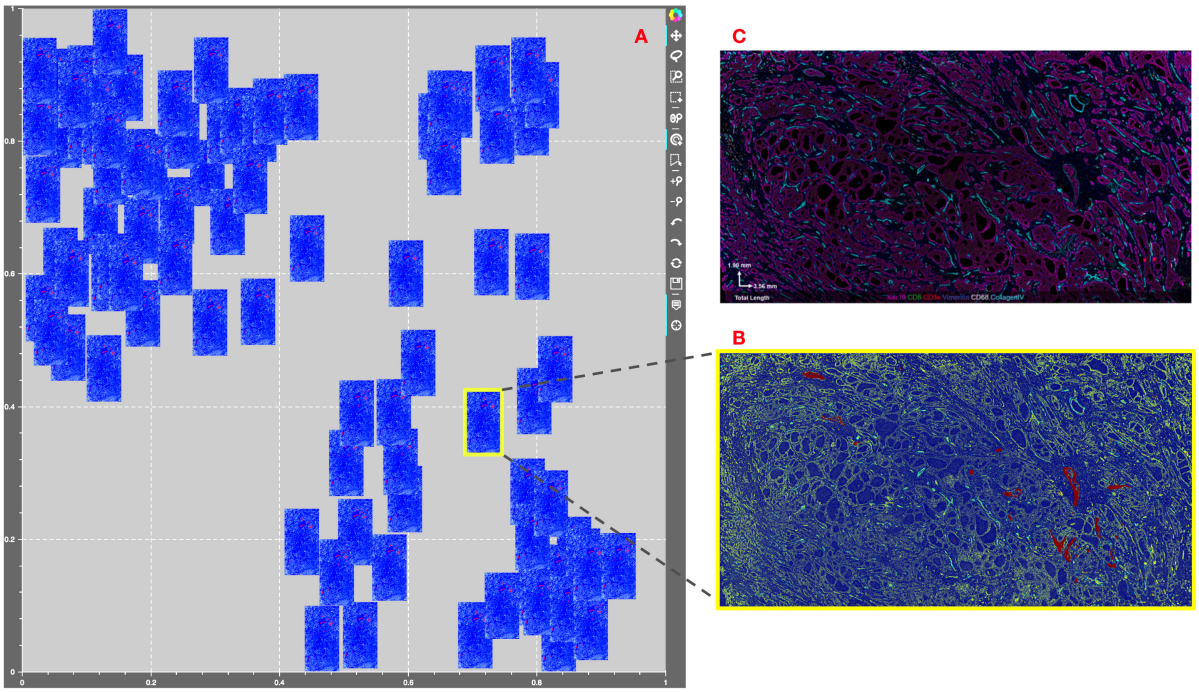


Figure showing Mistic tested on the 64-channel Human FFPE Breast adenocarcinoma CODEX data. **A.** The static canvas shows 88 repeats of the CODEX image arranged in proxy user-generated tSNE co-ordinates. Mistic gives the composite image using 7 markers (Keratin14, FoxP3, CD34, CD8, CD3e, CD68 and Perlecan). **B.** The zoomed in composite thumbnail generated by Mistic and **C.** The CODEX image for Keratin19, CD8, CD3e, Vimentin, CD68 and CollagenIV. Note that with Mistic we can view any number of markers at once.

5.8 Human FFPE Tonsil demo data from Akoya

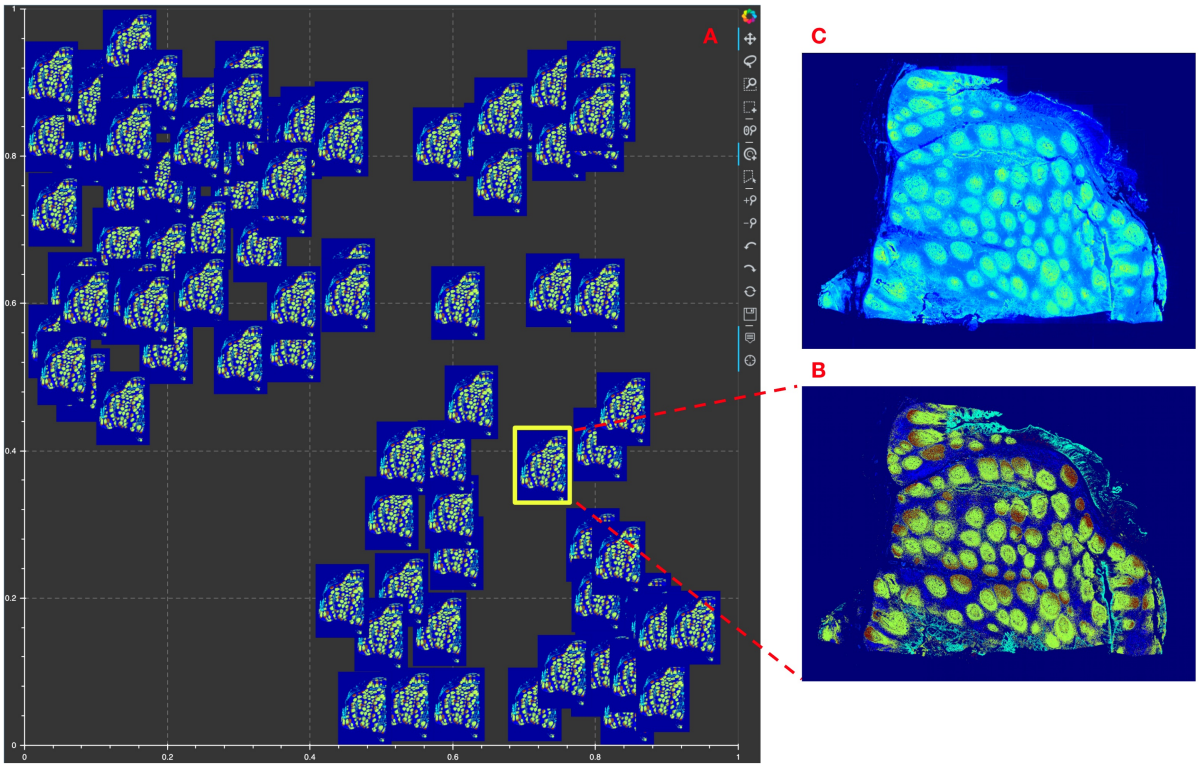


Figure showing Mistic tested on the 16-channel Human FFPE Tonsil CODEX pyramidal TIFF (QPTIFF) data provided to us by Akoya Biosciences. Note that we read in all the five image pyramids in the file and use the fourth image pyramid for rendering in Mistic. **A.** The static canvas shows 105 repeats of the CODEX image arranged in proxy user-generated tSNE co-ordinates. Mistic gives the composite image using 6 markers (PanCK, CD31, SMA, Ki67, CD8 and CD20). **B.** The zoomed in composite thumbnail generated by Mistic and **C.** The CODEX image for PanCK, CD31, Ki67 and CD20 rendered using QuPath [21]. Note that with Mistic we can view any number of markers at once.

5.9 Human Colorectal carcinoma

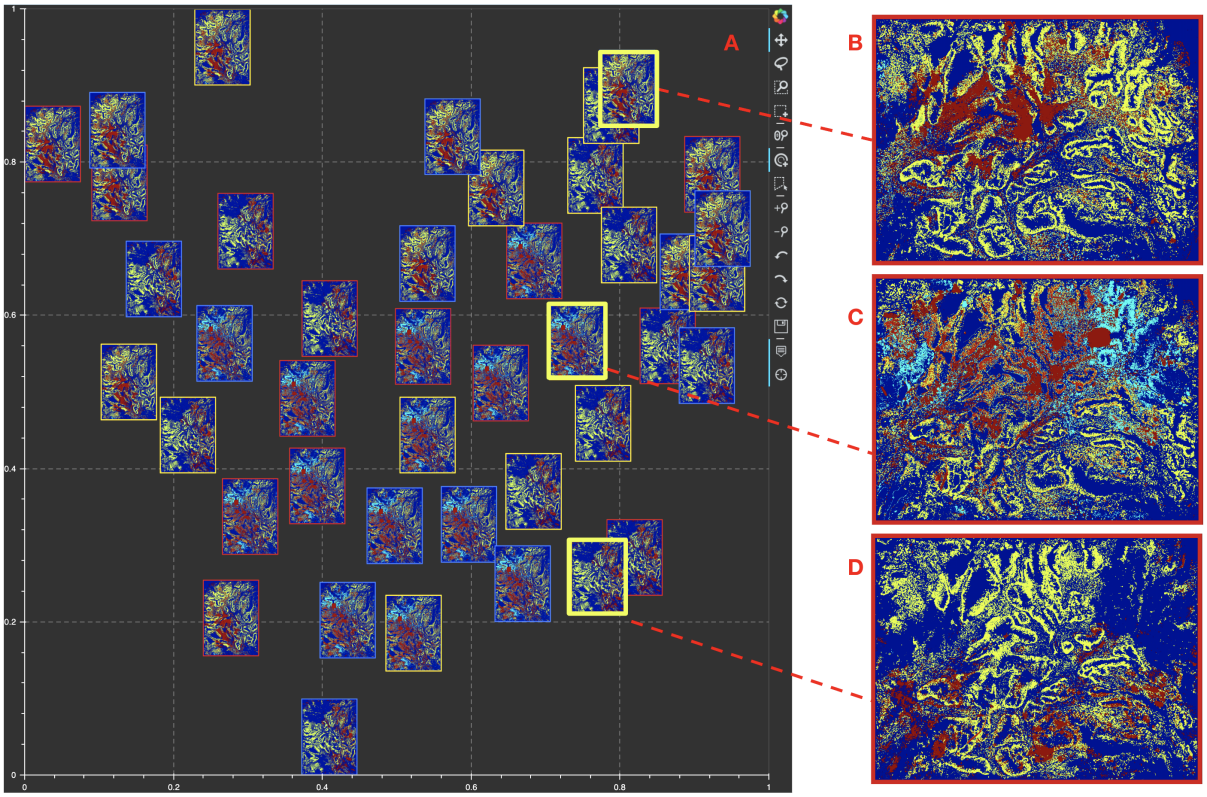


Figure showing Mistic tested on 4-channel Human colorectal carcinoma CyCIF (OME-TIFF) images [53]. **A.** The static canvas shows 42 duplicates of the three CyCIF images (DAPI, CD3, CD4, CD8, CD20, CD68, FoxP3) are rendered randomly. **B-D.** The zoomed-in composite thumbnails generated by Mistic for the three CyCIF images.

CHAPTER

SIX

VIGNETTES ON T-CYCIF DATA

We provide example vignettes to help users run Mistic. Check out our *Mistic experiments* page for examples Mistic has been tested on and *Functions* for a full list of functions Mistic uses.

6.1 t-CyCIF image on Lung adenocarcinoma metastasis to lymph node

This t-CyCIF image is in OME-TIFF format, 13GB in size with dimensions 10101 x 9666, and has 44 marker channels. To simultaneously test Mistic for scalability, we created duplicates of this image. In *Mistic experiments* 5.3, we use 70 duplicates of this image and render it on Mistic for 6 markers: CD45, Keratin, aSMA, FoXP3, PD-1, PD-L1.

- Download Mistic as described here *Code Installation*
  - Navigate to `Mistic_code/code/user_inputs/`
- Download data from link.
- Go to Files -> DATASET-1 -> LUNG-1-LN -> 40X-> ometiff.
- Download `lung-1-ln_40x.ome.tif` and place it in the /user_inputs/figures/ folder
- Create say 3 copies of this .tif file in the /figures folder
- Navigate to the /user_inputs/metadata folder
  - Upload the imaging markers of interest as Markers_ids.csv
    - A sample .csv is provided in the /metadata/t-CyCIF folder
    - If using the sample .csv, move this file from /metadata/t-CyCIF to /metadata folder
  - Upload the markers.csv provided by t-CyCIF
    - Go to link.
    - Go to Files -> DATASET-1 -> markers.csv
  - Optional uploads:
    - Image tSNE co-ordinates as X_imagetSNE.csv
      - If no user-generated tSNE co-ordinates are provided, Mistic will generate a set of t-SNE coordinates to render the images
    - Cluster labels as Cluster_categories.csv
      - If cluster labels are not provided, Mistic will cluster the images using a Bayesian mixture model.

        * Patient_ids as Patient_ids.csv
        * Treatments as Treatment_catgories.csv
        * Patient response as Response_categories.csv
        * If any of these are unavailable, Mistic will use either the randomly-generated or user-provided tSNE points without any color coding i.e. dots are colored in gray, for the live panels.
        * Sample metadata files are provided for reference in the /metadata/t-CyCIF folder
        * If using the sample metadata, move the files from /metadata/t-CyCIF/ into the /metadata folder

- Open a command prompt (or the Terminal application), change to the directory containing /code and type
    - `bash mistic.sh`
    - This runs a bokeh server locally and will automatically open the interactive dashboard in your browser at http://localhost:5098/image_tSNE_GUI
- From the GUI, choose 't-CyCIF' from the dropdown menu, check the boxes for markers of interest and select other options based on image metadata (for example, border, image layout) and click 'Run'

6.2 t-CyCIF image on Primary lung squamous cell carcinoma

This is an example where Mistic can be used to view a stack montage made up of the individual markers for a single multiplexed image (see *Mistic experiments* 5.4). We show this option on the t-CyCIF image on Primary lung squamous cell carcinoma. This is in OME-TIFF format for all 44 marker channels.

- Download Mistic as described here *Code Installation*
    - Navigate to `Mistic_code/code/user_inputs/`
- Download data from link.
    - Go to Files -> DATASET-1 -> LUNG-3-PR -> 40X-> singletiff.
    - Download the 44 single channel tifs (`lung-3-pr_40x_x.tif`) and place it in the /user_inputs/figures/ folder
- Navigate to the /user_inputs/metadata folder
    - Upload the markers.csv provided by t-CyCIF
        * Go to link.
        * Go to Files -> DATASET-1 -> markers.csv
- Open a command prompt (or the Terminal application), change to the directory containing /code and type
    - `bash mistic.sh`
    - This runs a bokeh server locally and will automatically open the interactive dashboard in your browser at http://localhost:5098/image_tSNE_GUI
- From the GUI, choose 't-CyCIF' from the dropdown menu, select the 'Stack montage' option and click 'Run'

CHAPTER

SEVEN

VIGNETTES ON VECTRA DATA

We provide a toy dataset to run Mistic on the NSCLC Vectra FoVs. Check out our *Mistic experiments* page for examples Mistic has been tested on and *Functions* for a full list of functions Mistic uses.

7.1 PerkinElmer Vectra based NSCLC FoVs

Each Vectra image is in TIFF format, is up to 43MB in size with dimensions 1344 x 1008, and has 7 marker channels.

- Download Mistic as described here *Code Installation*
  - Navigate to `Mistic_code/code/user_inputs/`
- Download data from link.
- Place it in the /user_inputs/figures/ folder
- Navigate to the /user_inputs/metadata folder
  - Upload the imaging markers of interest as Markers_ids.csv
    - A sample .csv is provided in the /metadata/Vectra folder
    - If using the sample .csv, move this file from /metadata/Vectra to /metadata folder
  - Upload the markers.csv which is the entire set of markers for that dataset
    - A sample .csv is provided in the /metadata/Vectra folder
    - If using the sample .csv, move this file from /metadata/Vectra to /metadata folder
  - Optional uploads:
    - Image tSNE co-ordinates as X_imagetSNE.csv
      - If no user-generated tSNE co-ordinates are provided, Mistic will generate a set of random coordinates to render the images
    - Cluster labels as Cluster_categories.csv
      - If cluster labels are not provided, Mistic will cluster the images using a Bayesian mixture model.
    - Patient_ids as Patient_ids.csv
    - Treatments as Treatment_catgories.csv
    - Patient response as Response_categories.csv
    - If any of these are unavailable, Mistic will use either the randomly-generated or user-provided tSNE points without any color coding i.e. dots are colored in gray, for the live panels.
    - Sample metadata files are provided for reference in the /metadata/Vectra folder

        * If using the sample metadata, move the files from /metadata/Vectra into the /metadata folder

- Open a command prompt (or the Terminal application), change to the directory containing /code and type
  - `bash mistic.sh`
  - This runs a bokeh server locally and will automatically open the interactive dashboard in your browser at http://localhost:5098/image_tSNE_GUI
- From the GUI, choose 'Vectra' from the dropdown menu, check the boxes for markers of interest and select other options based on image metadata (for example, border, image layout) and click 'Run'

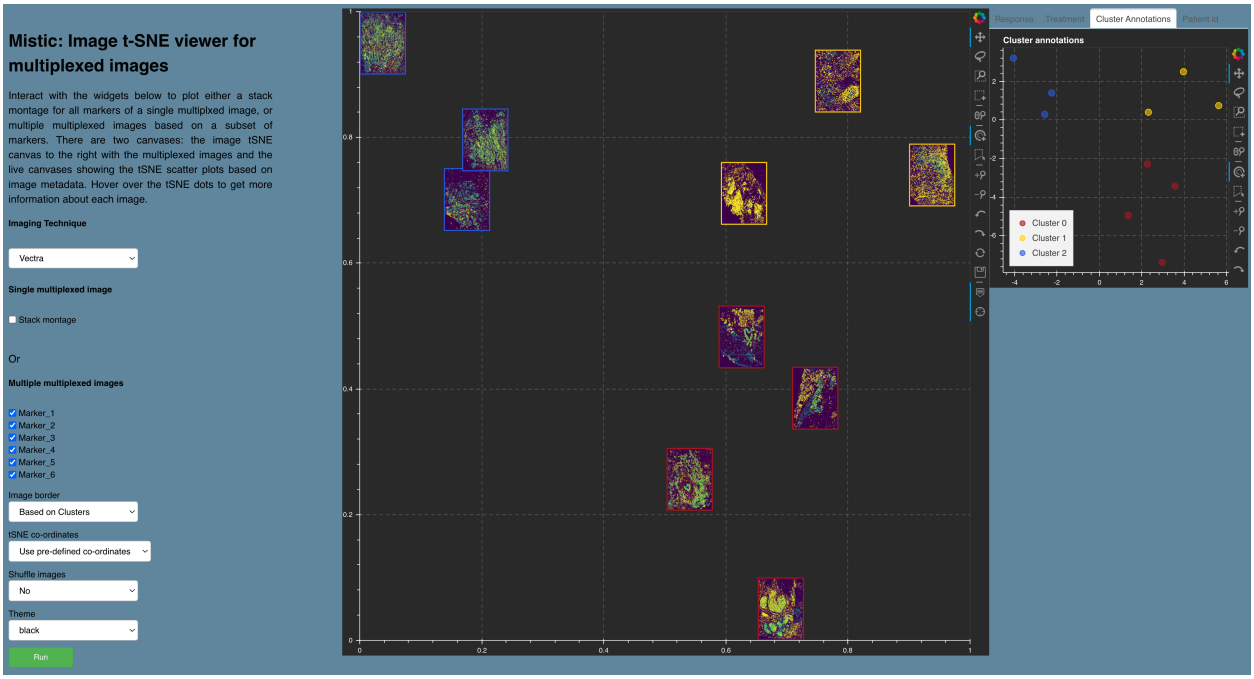


Figure showing Mistic's output on toy data based on user inputs. The static canvas renders the image t-SNE and the live canvas shows the t-SNE scatter plot for cluster annotations. Each image border is colored based on cluster identity.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search