ETH zürich

Department of Biosystems Science and Engineering

Pseudo-temporal ordering and RNA velocity – Practical session

Zhisong He, PhD
Senior researcher and Lecturer
Treutlein lab, D-BSSE, ETH Zurich
Basel, Switzerland
11.07.2024

Workshop

The Hitchhiker's Guide to scRNA-seq

Outlines

- **Introduction**
  - Setup your environment (R & Python)
  - The example data set

- **Analysis**
  - Brief overview on the data (R)
  - Diffusion pseudotime and trajectory analysis (R)
  - Data conversion (R & Python)
  - Diffusion pseudotime (Python)
  - Coarse-grained trajectory analysis with PAGA (Python)
  - RNA velocity analysis with scVelo (Python)
  - Fate probability estimation with CellRank 2 (Python)

Set up your conda environment

Packages in need:
- R
  - Seurat
  - reticulate
  - anndata
  - destiny
  - URD
- Python
  - scanpy
  - scvelo
  - cellrank
  - (velocyto.py)

Set up your conda environment (Linux/MacOS users):

```
> conda create -n env_hitchhiker2024 python=3.9 r-base=4 jupyterlab r-reticulate r-irkernel r-devtools
scanpy scvelo cellrank python-igraph r-Seurat=5 cython gsl udunits2 -c conda-forge --solver=libmamba
> conda activate env_hitchhiker2024
> conda install -c bioconda -c conda-forge velocyto.py r-anndata
(Linux) > conda install -c conda-forge gcc gxx
(Linux) > conda install -c bioconda -c conda-forge bioconductor-destiny
(MacOS/Linux-failed) > conda install -c conda-forge r-biocmanager cmake
(MacOS/Linux-failed) > echo 'BiocManager::install("destiny")' | R --vanilla
> echo 'devtools::install_github("farrellja/URD")' | R --vanilla
> echo 'devtools::install_github("mojaveazure/seurat-disk")' | R --vanilla
```

R package compilation environment for Win users
https://cran.r-project.org/bin/windows/Rtools/

Alternative option for Win users
Install a Linux environment with WSL2

Compilation tools for MacOS users
https://mac.r-project.org/tools/

More for MacOS (ARM64) users:
At terminal, do

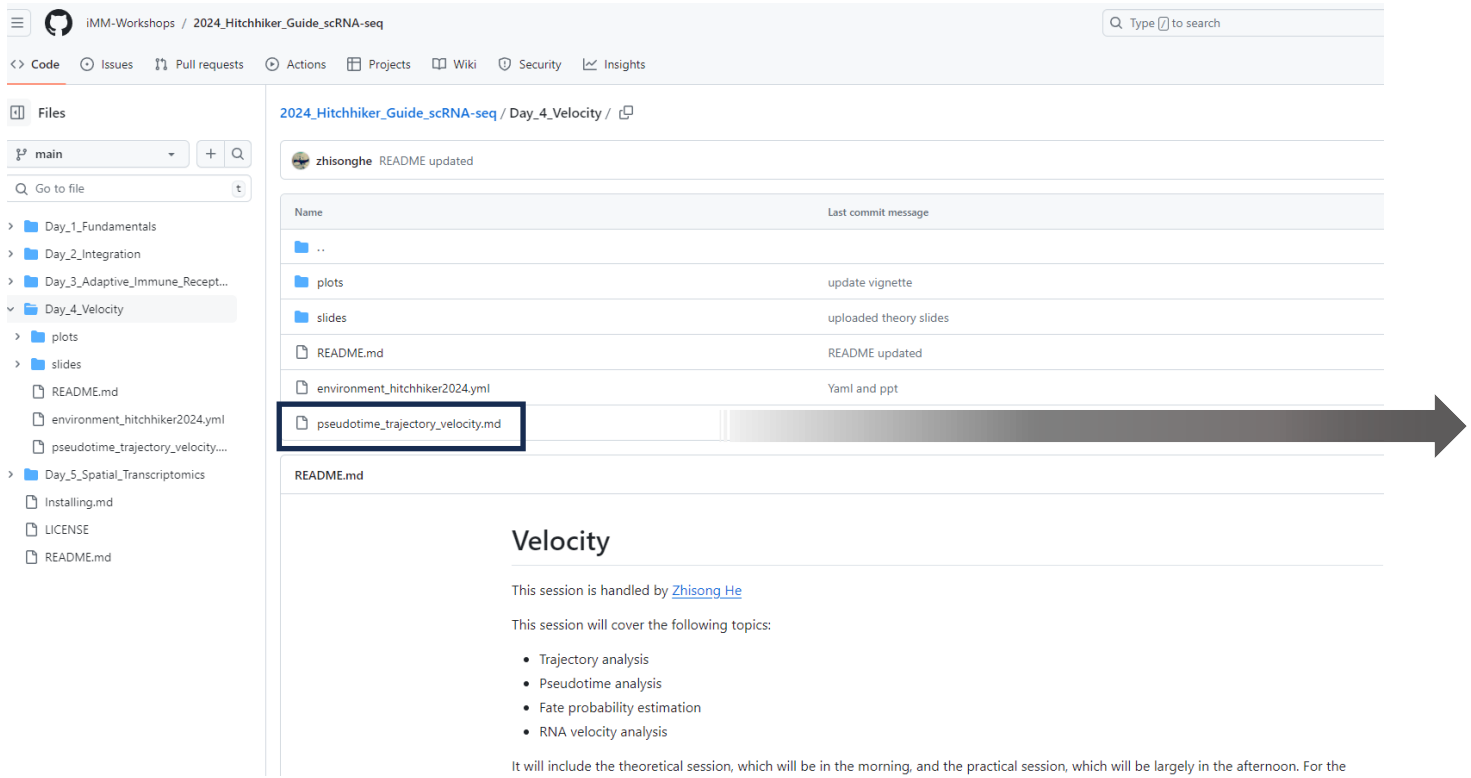
```
> conda activate env_hitchhiker2024
> open `which R | sed 's/bin\/R$/lib\/R\/etc\/Makeconf/'`
```

This will open the TextEdit app. Look for the line starting with CPPFLAGS. Add the following content to the end of the line: `-DHAVE_WORKING_LOG1P`
Next, save and close the file

More information can be seen here:
https://github.com/iMM-Workshops/2024_Hitchhiker_Guide_scRNA-seq/blob/main/Day_4_Velocity/README.md
(Also includes non-conda way of setting up the environment)

Online vignette for the analysis

https://github.com/iMM-Workshops/2024_Hitchhiker_Guide_scRNA-seq/blob/main/Day_4_Velocity/



(R) Load the data and check information

Load the Seurat object and read in the object

```
library(Seurat)
seurat <- readRDS('DS1.rds')
```

```
Loading required package: SeuratObject
Loading required package: sp

Attaching package: 'SeuratObject'

The following object is masked from 'package:base':

    intersect
```

Check the basic information of the data

```
dim(seurat)
head(seurat@meta.data)
names(seurat@reductions)
```

<style> .list-inline {list-style: none; margin:0; padding: 0} .list-inline>li {display: inline-block} .list-inline>li:not(:last-child):after {content: "\00b7"; padding: 0 .5ex} </style>
1. 33694
2. 4317

A data.frame: 6 x 6

	orig.ident	nCount_RNA	nFeature_RNA	percent.mt	celltype	region
	<fct>	<dbl>	<int>	<dbl>	<fct>	<fct>
AAACCTGAGAATGTTG-1	DS1	16752	3287	1.098376	G2M Dien. and midbrain NPC	Non-telencephalon
AAACCTGAGAGCCTAG-1	DS1	13533	3399	1.950787	G2M Dien. and midbrain NPC	Non-telencephalon
AAACCTGAGTAATCCC-1	DS1	3098	1558	2.453196	Midbrain-hindbrain boundary neuron	Non-telencephalon
AAACCTGCACACTGCG-1	DS1	5158	2015	3.761148	Midbrain-hindbrain boundary neuron	Non-telencephalon
AAACCTGCATCGGAAG-1	DS1	6966	2322	2.182027	Dorsal telen. IP	Dorsal telencephalon
AAACCTGGTGTAACGG-1	DS1	4108	1542	2.507303	Unknown 1	NA

<style> .list-inline {list-style: none; margin:0; padding: 0} .list-inline>li {display: inline-block} .list-inline>li:not(:last-child):after {content: "\00b7"; padding: 0 .5ex} </style>
1. 'pca'
2. 'umap'
3. 'tsne'

Feature plots to better understand the data

Alternative vignette for the analysis (R/Seurat-centric)

https://github.com/quadbio/scRNAseq_analysis_vignette

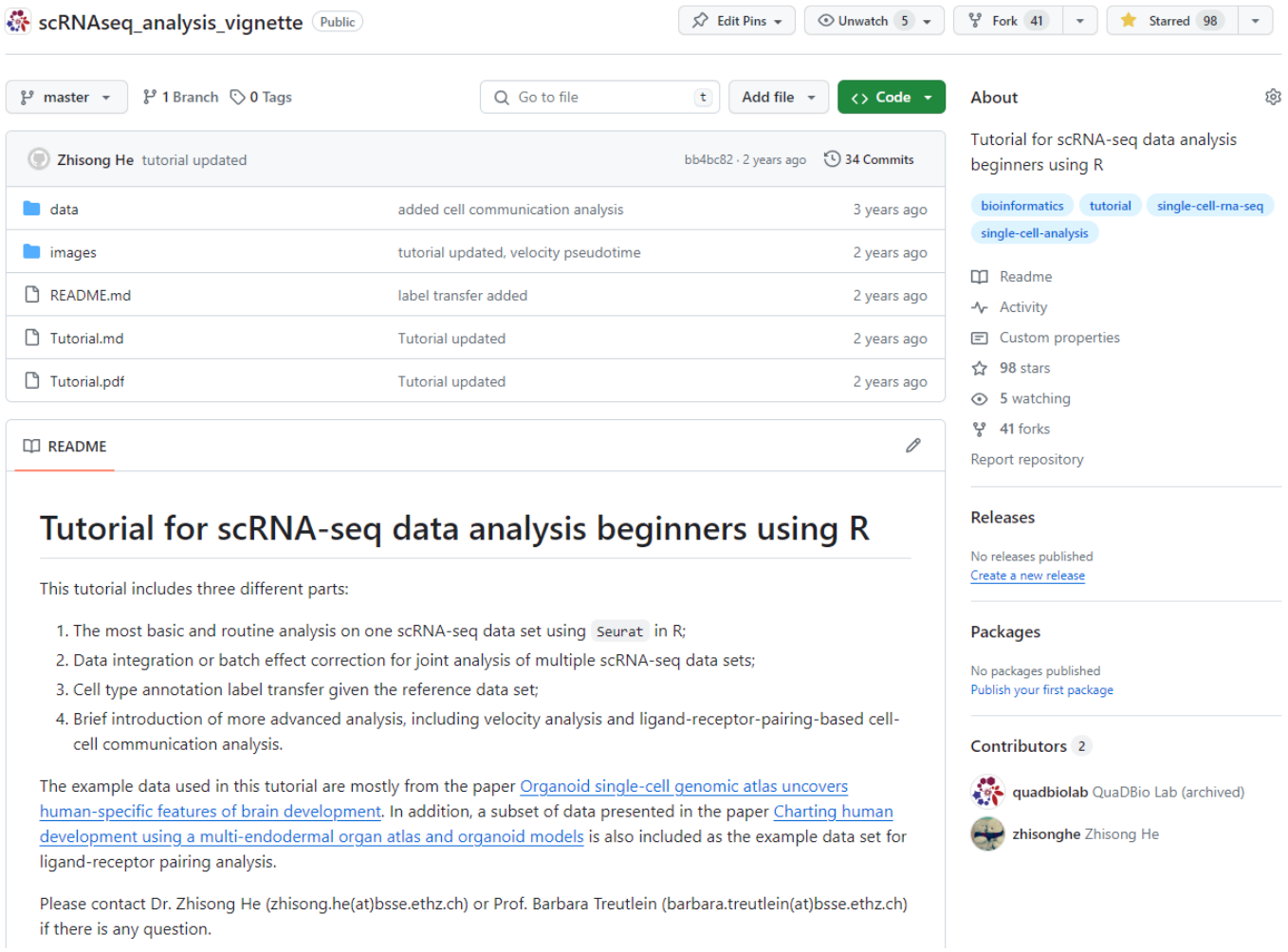


Table of Content

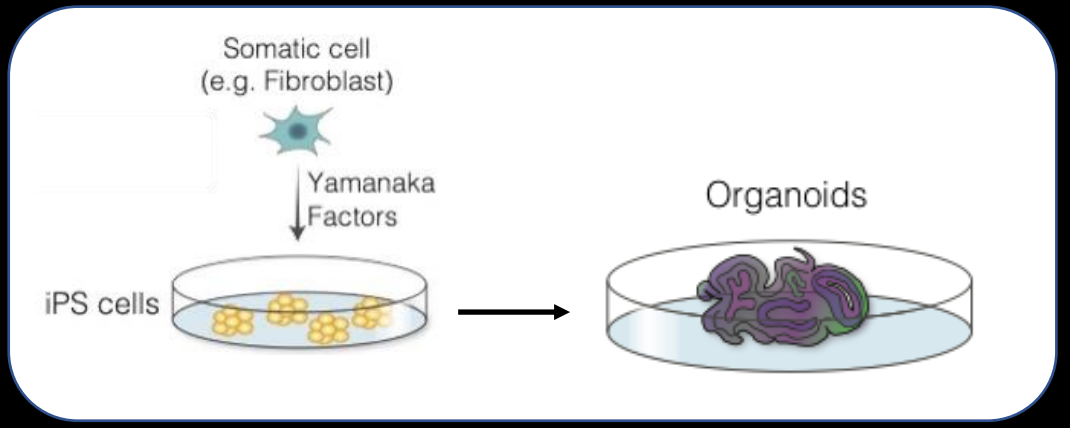
Example data set: scRNA-seq data of brain organoids

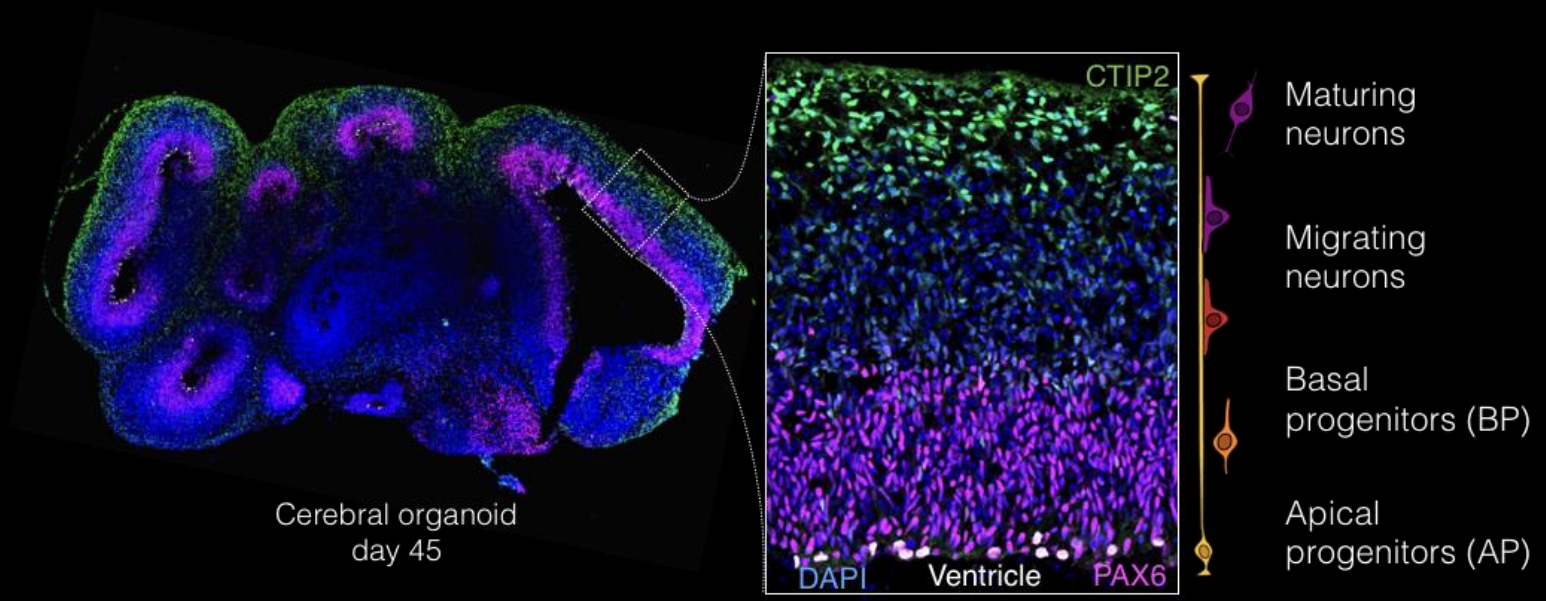
LETTER

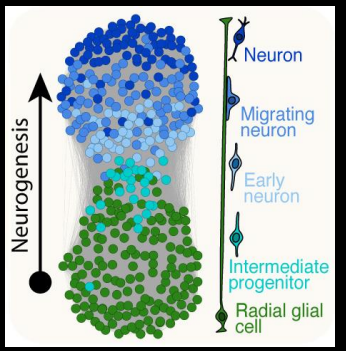
https://doi.org/10.1038/s41586-019-1654-9

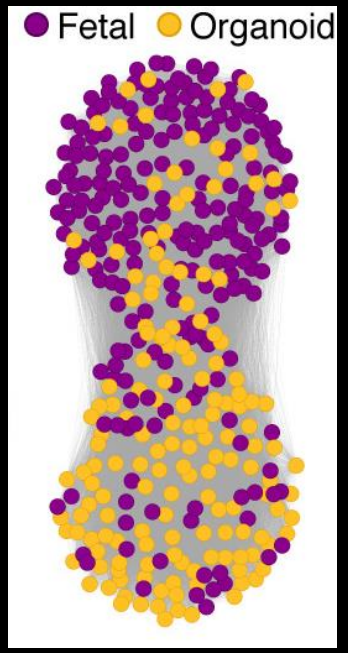
Organoid single-cell genomic atlas uncovers human-specific features of brain development

Sabina Kanton[1,7], Michael James Boyle[1,7], Zhisong He[1,2,7]*, Malgorzata Santel[1], Anne Weigert[1], Fátima Sanchís-Calleja[1,2], Patricia Guijarro[3], Leila Sidow[1], Jonas Simon Fleck[2], Dingding Han[3], Zhengzong Qian[3], Michael Heide[4], Wieland B. Huttner[4], Philipp Khaitovich[1,3,5], Svante Pääbo[1], Barbara Treutlein[1,2]* & J. Gray Camp[1,6]*



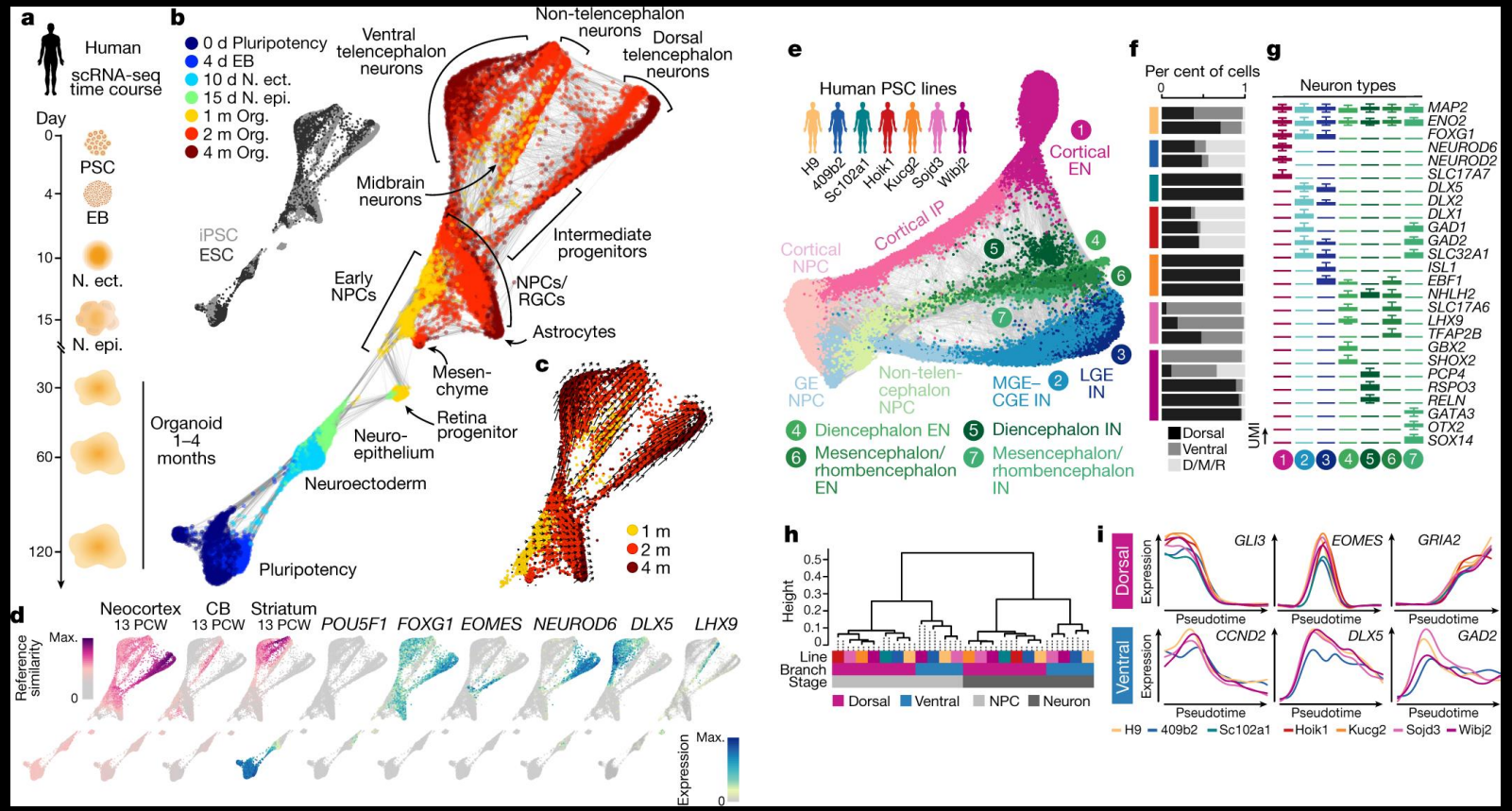


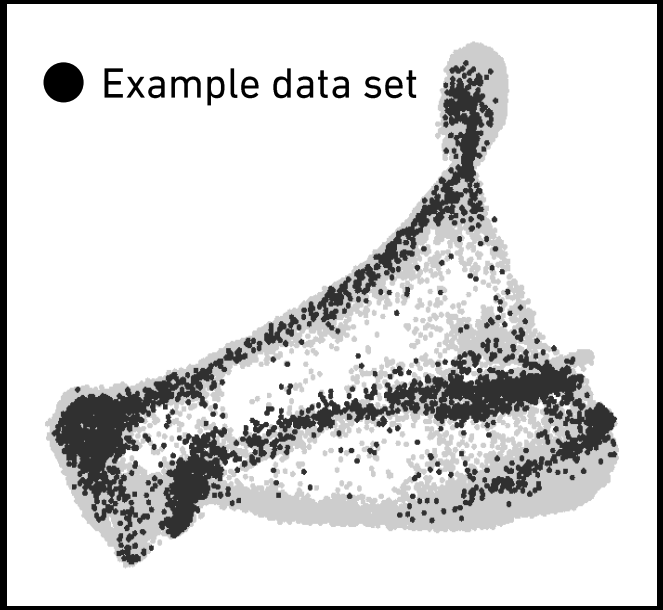




Camp et al. (2015) Human cerebral organoids recapitulate gene expression programs of fetal neocortex development. *PNAS*

Example data set: scRNA-seq data of brain organoids





Example data set

One 2-month-old organoid

- 4317 cells
- Seurat object
- Preprocessed and annotated
- Also include exonic/intronic count matrices as additional assays

Link to the data (RDS file for the Seurat object):
https://polybox.ethz.ch/index.php/s/bjNnfD9I3rwpjIt

Link to the data (H5AD file for the AnnData object):
https://polybox.ethz.ch/index.php/s/bUYZE6qPgROBggH

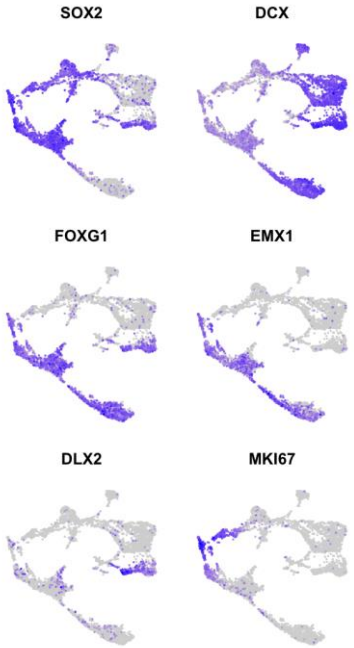
Outlines

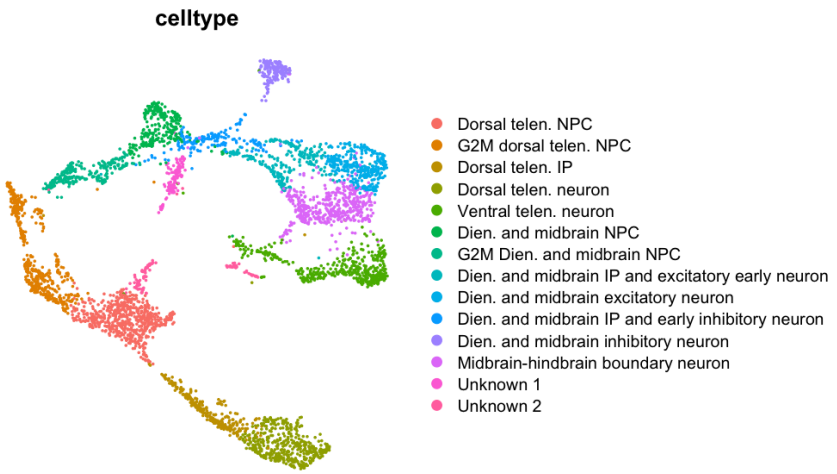
- **Introduction**
  - Setup your environment (R & Python)
  - The example data set

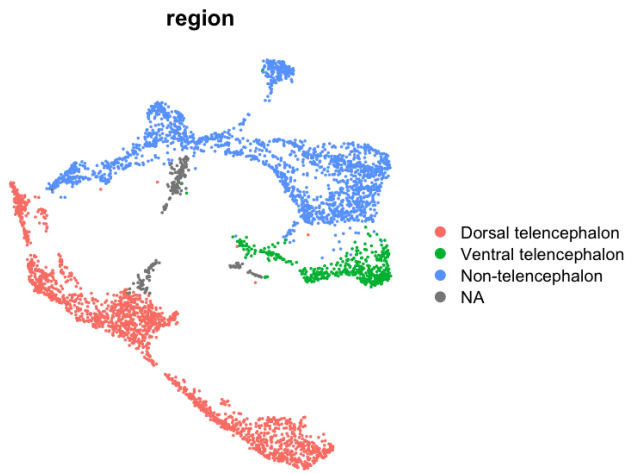
- **Analysis**
  - Brief overview on the data (R) – 5 min
  - Diffusion pseudotime and trajectory analysis (R) – 20 min
  - Data conversion (R & Python) – 5 min
  - Diffusion pseudotime (Python) – 10 min
  - Coarse-grained trajectory analysis with PAGA (Python) – 10 min
  - RNA velocity analysis with scVelo (Python) – 20 min
  - Fate probability estimation with CellRank 2 (Python) – 20 min

Brief overview on the data

```
> library(Seurat)

> seurat <- readRDS('DS1.rds')

> dim(seurat)
> head(seurat@meta.data)
> names(seurat@reductions)

> FeaturePlot(seurat, c('SOX2','DCX','FOXG1','EMX1','DLX2','MKI67'), order=T) & NoAxes() & NoLegend()
> p1 <- UMAPPlot(seurat, group.by=c('celltype')) & NoAxes()
> p2 <- UMAPPlot(seurat, group.by=c('region')) & NoAxes()
> p1 | p2
```



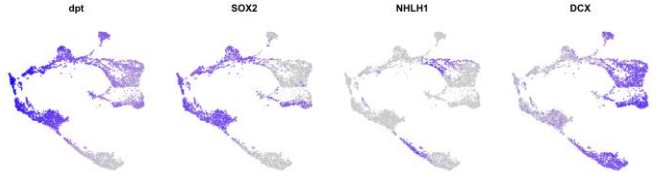




Diffusion map in R

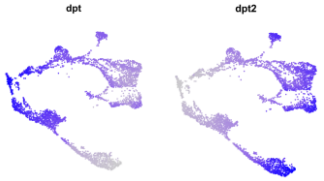
```
> library(density)

> seurat <- subset(seurat, subset = celltype %in% setdiff(levels(seurat$celltype),
c('Unknown 1','Unknown 2')))
> seurat <- RunPCA(seurat, npcs=20)
> dm <- DiffusionMap(Embeddings(seurat, "pca")[,1:20], k=50)
> dpt <- DPT(dm)
> seurat$dpt <- rank(dpt$dpt)
> FeaturePlot(seurat, c("dpt","SOX2","NHLH1","DCX"), ncol=4) & NoAxes() & NoLegend()
```

Run DPT with default parameters




```
> tips_cand <- sapply(1:100, function(i){ random_root(dm) })
> idx_NPC <- which(seurat@meta.data$celltype %in% c('Dorsal telen. NPC',
                                                    'G2M dorsal telen. NPC',
                                                    'Dien. and midbrain NPC',
                                                    'G2M Dien. and midbrain NPC'))
> tips_cand <- as.numeric(names(which.max(table(tips_cand[tips_cand %in% idx_NPC]))))
> dpt2 <- DPT(dm, tips=tips_cand)
> seurat$dpt2 <- rank(dpt2$dpt)

> FeaturePlot(seurat, c("dpt","dpt2"), ncol=2) & NoAxes() & NoLegend()
```

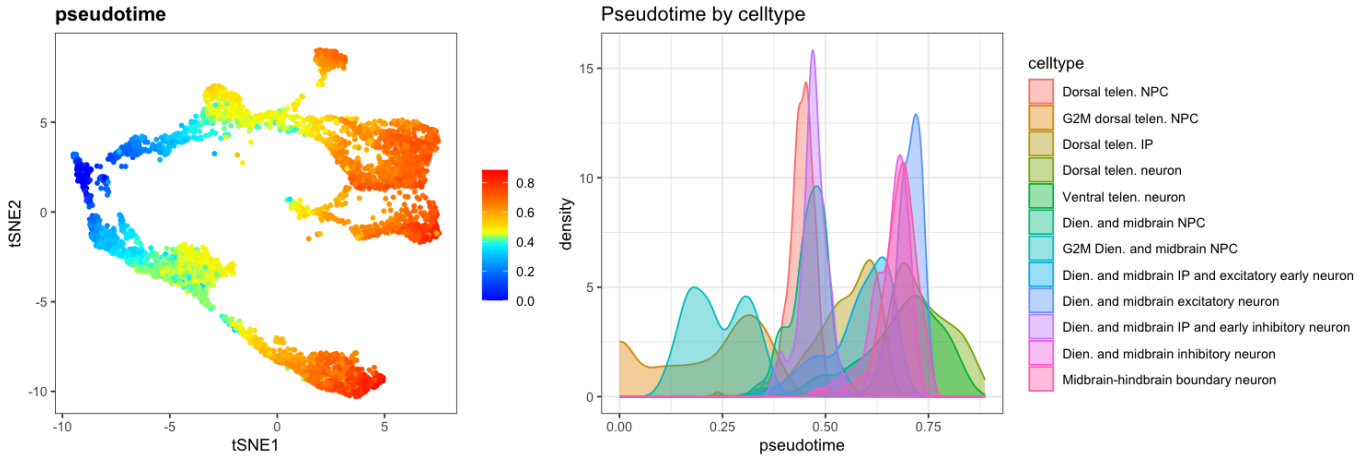
Run DPT with default parameters, while ensuring
a progenitor cell is chosen as the tip



(Optional) Trajectory analysis in R with URD

```
> library(URD)

> urd <- createURD(count.data = seurat[['RNA']]@counts, meta=seurat@meta.data,
min.cells=0, min.counts=0)
> urd@pca.scores <- as.data.frame(Embeddings(seurat,'pca'))
> urd@tsne.y <- setNames(as.data.frame(Embeddings(seurat,'umap')), c('tSNE1','tSNE2'))
> urd@dm <- dm
> root_cells <- colnames(seurat)[order(seurat$dpt2)[1:100]]
> floods <- floodPseudotime(urd, root.cells = root_cells, n=50, minimum.cells.flooded =
2, verbose=F)
> urd <- floodPseudotimeProcess(urd, floods, floods.name="pseudotime")
> root_cells <- rownames(urd@meta)[order(urd@pseudotime$pseudotime)[1:50]]
> p1 <- plotDim(urd, "pseudotime")
> p2 <- plotDists(urd, "pseudotime", "celltype", plot.title="Pseudotime by celltype")
> p1  | p2
```

Get flood diffusion based pseudotime with URD



(Optional) Trajectory analysis in R with URD (2)

```
> seurat@meta.data$fpt <- urd@pseudotime$pseudotime
> cor(seurat$fpt, seurat$dpt2, method='spearman')
> plot(seurat$fpt, seurat$dpt2, pch=16, col='#30303050', frame=F)
```

Compare URD-based pseudotime with DPT

```
> neuron_types <- setdiff(grep('neuron', levels(seurat$celltype), value=T), grep('early',
levels(seurat$celltype), value=T))
> idx_tips <- unlist(lapply(neuron_types, function(x){
    which(seurat$celltype==x)[order(seurat$fpt[seurat$celltype == x], decreasing=T)[1:50]]
  }))
> urd@group.ids[idx_tips, 'tip.clusters'] <- as.numeric(droplevels(seurat$celltype[idx_tips]))
```

Assign tip populations

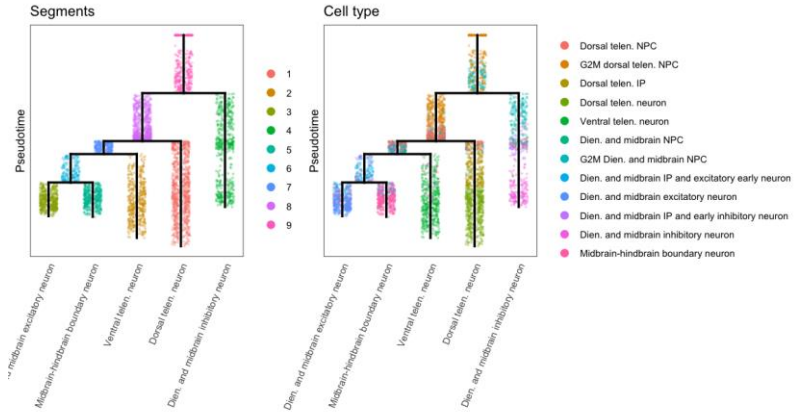
```
> ptlogistic <- pseudotimeDetermineLogistic(urd, "pseudotime", optimal.cells.forward=20,
max.cells.back=20, do.plot = T)
> biased.tm <- as.matrix(pseudotimeWeightTransitionMatrix(urd, "pseudotime",
logistic.params=ptlogistic))
> walks <- simulateRandomWalksFromTips(urd, tip.group.id = "tip.clusters", root.cells = root_cells,
transition.matrix = biased.tm, n.per.tip = 5000, root.visits = 1, max.steps = 4000, verbose = F)
> urd <- processRandomWalksFromTips(urd, walks, verbose = F)
```

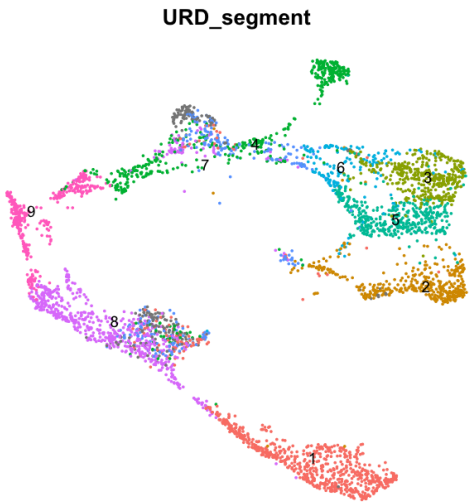
Random walk from roots to tips

(Optional) Trajectory analysis in R with URD (3)

```
> tree <- loadTipCells(urd, "tip.clusters")
> tree <- buildTree(tree, pseudotime = "pseudotime", tips.use=NULL, divergence.method = "preference",
cells.per.pseudotime.bin = 25, bins.per.pseudotime.window = 8, save.all.breakpoint.info = T,
p.thresh=0.001)
> tree <- nameSegments(tree, segments= sort(unique(tree@group.ids$tip.clusters)), segment.names =
levels(droplevels(seurat$celltype[idx_tips])), short.names = c('dTN','vTN','DMExN','DMInN','MHBN'))
>
> p1 <- plotTree(tree, "segment", title="Segments")
> p2 <- plotTree(tree, "celltype", title="Cell type")
> p1 | p2
```

Build and visualize the differentiation tree






```
> seurat@meta.data$URD_segment <- tree@group.ids$segment
> UMAPPlot(seurat, group.by='URD_segment', label=T) & NoAxes() & NoLegend()
```

Data conversion from Seurat to AnnData (h5ad)

```
> library(anndata)
> library(Matrix)
> shared_genes <- intersect(rownames(seurat[['RNA']]),
                            intersect(rownames(seurat[['spliced']]),
                                      rownames(seurat[['unspliced']])))
> adata <- AnnData(X = t(seurat[['RNA']]@data[shared_genes,]),
                   obs = seurat@meta.data,
                   var = seurat[['RNA']]@meta.features[shared_genes,],
                   layers = list(counts = t(seurat[['RNA']]@counts[shared_genes,]),
                                 spliced = t(seurat[['spliced']]@counts[shared_genes,]),
                                 unspliced = t(seurat[['unspliced']]@counts[shared_genes,])),
                   obsm = list(X_pca = Embeddings(seurat,"pca")[,1:20],
                               X_umap = Embeddings(seurat,"umap"))
                  )
> adata$write_h5ad("DS1.h5ad")
```

P.S. *SeuratDisk* (https://github.com/mojaveazure/seurat-disk) also provides the Seurat to h5ad conversion functionality. However, it designs to work for only one Assay, for which it converts the "*data*" slot/layer of the Array into the "*X*" slot of the AnnData, and the "*counts*" slot/layer into a matrix in the "*layers*" slot of the AnnData. This doesn't work for what we want to do here

P.S. In an AnnData object, it is required that all data matrices (*X* and all matrices in the "*layer*" slot) share the same dimensionalities. Therefore, we have to subset into genes appear in all the three matrices.

Diffusion map in Python

```
>>> import scanpy as sc
>>> adata = sc.read_h5ad('DS1.h5ad')
>>> sc.pp.neighbors(adata, n_neighbors=50, n_pcs=20, use_rep='X_pca')
>>> sc.tl.diffmap(adata, n_comps=20)
```

Run diffusion map

P.S. In *scanpy*, the diffusion pseudotime (*scanpy.tl.dpt*) function requires the root cell to be manually labeled. To use the same root guessing procedure as implemented in *destiny* in R, we have to re-implement it with the following code:

```
import random
import numpy as np
import pandas as pd
def random_root(adata, seed = None, neighbors_key=None, idx_subset = None):
    if seed is not None:
        random.seed(seed)
    iroot_bak = None
    if 'iroot' in adata.uns.keys():
        iroot_bak = adata.uns['iroot'].copy()
    dpt_bak = None
    if 'dpt_pseudotime' in adata.obs.columns:
        dpt_bak = adata.obs['dpt_pseudotime'].copy()

    idx = np.random.choice(list(range(adata.shape[0])))
    adata.uns['iroot'] = idx
    sc.tl.dpt(adata, neighbors_key=neighbors_key)
    dpt = adata.obs['dpt_pseudotime']
    if idx_subset is not None:
        dpt = dpt.iloc[idx_subset]
    idx_max_dpt = np.argmax(dpt)
    if idx_subset is not None:
        idx_max_dpt = idx_subset[idx_max_dpt]

    del adata.uns['iroot']
    del adata.obs['dpt_pseudotime']
    if iroot_bak is not None:
        adata.uns['iroot'] = iroot_bak.copy()
    if dpt_bak is not None:
        adata.obs['dpt_pseudotime'] = dpt_bak.copy()

    return idx_max_dpt
```

Diffusion map in Python (2)

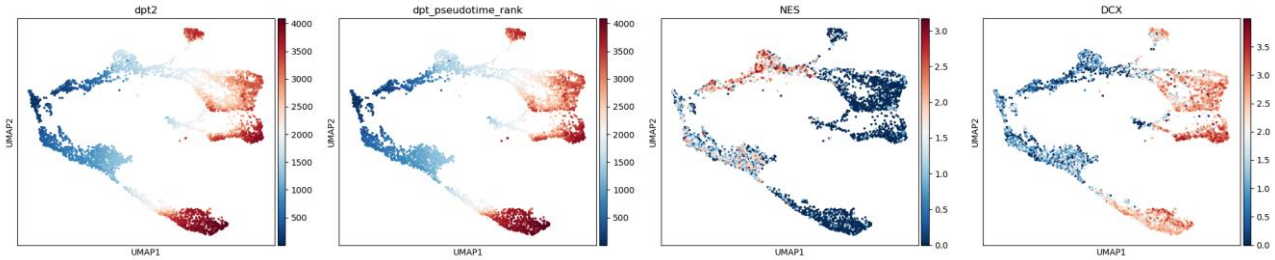
```
>>> idx_subset = np.where(np.isin(adata.obs['celltype'], ['Dorsal telen. NPC',
                                                          'G2M dorsal telen. NPC',
                                                          'Dien. and midbrain NPC',
                                                          'G2M Dien. and midbrain NPC']))[0]
>>> idxs_rand_root = np.apply_along_axis(lambda x: random_root(adata, idx_subset=idx_subset),
                                         1, np.array(range(1000))[:,None])
>>> adata.uns['iroot'] = np.argmax(np.bincount(idxs_rand_root))
>>> sc.tl.dpt(adata, n_dcs=20)
```

Infer root cell among NPCs, and estimate diffusion pseudotimes

```
>>> from scipy.stats import pearsonr, spearmanr, rankdata
>>> [pearsonr(adata.obs['dpt2'], adata.obs['dpt_pseudotime']),
     spearmanr(adata.obs['dpt2'], adata.obs['dpt_pseudotime'])]

>>> adata.obs['dpt_pseudotime_rank'] = rankdata(adata.obs['dpt_pseudotime'])
>>> import matplotlib.pyplot as plt
>>> plt.scatter(adata.obs['dpt2'], adata.obs['dpt_pseudotime_rank'])
>>> plt.show()
```

Compare the Python-based DPT and R-based DPT

```
>>> sc.pl.umap(adata, color=['dpt2','dpt_pseudotime_rank','NES','DCX'], color_map='RdBu_r', ncols=4)
```

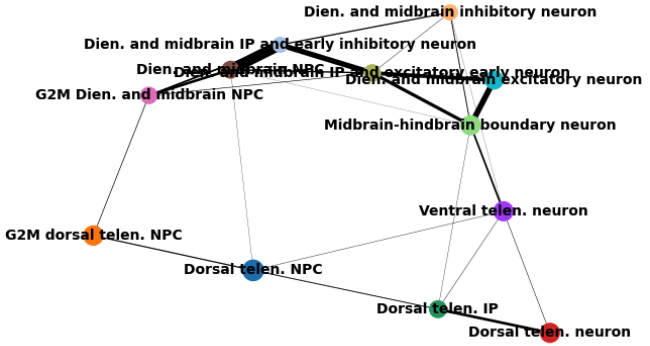


Coarse-grained trajectory analysis with PAGA

```
>>> adata.obs['celltype'] = adata.obs['celltype'].cat.remove_unused_categories()
>>> sc.pp.neighbors(adata, n_neighbors=20, n_pcs=20, use_rep='X_diffmap')
>>> sc.tl.paga(adata, groups='celltype')
>>> sc.pl.paga(adata)
```

Perform PAGA and visualize estimated cell type connectivities

	node1	node2
0	G2M dorsal telen. NPC	Dorsal telen. NPC
1	Dorsal telen. neuron	Dorsal telen. IP
2	G2M Dien. and midbrain NPC	Dien. and midbrain NPC
3	Dien. and midbrain IP and excitatory early neuron	Dien. and midbrain NPC
4	Dien. and midbrain excitatory neuron	Dien. and midbrain IP and excitatory early neuron
5	Dien. and midbrain IP and early inhibitory neuron	Dien. and midbrain NPC
6	Dien. and midbrain IP and early inhibitory neuron	G2M Dien. and midbrain NPC
7	Dien. and midbrain IP and early inhibitory neuron	Dien. and midbrain IP and excitatory early neuron
8	Dien. and midbrain inhibitory neuron	Dien. and midbrain IP and early inhibitory neuron
9	Midbrain-hindbrain boundary neuron	Ventral telen. neuron
10	Midbrain-hindbrain boundary neuron	Dien. and midbrain IP and excitatory early neuron
11	Midbrain-hindbrain boundary neuron	Dien. and midbrain excitatory neuron

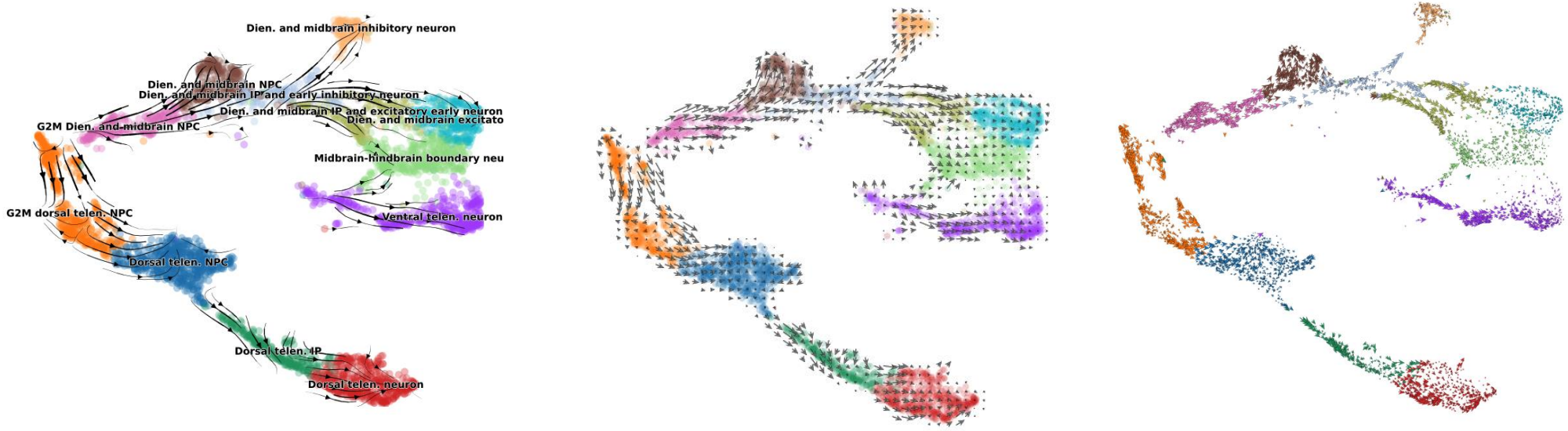



```
>>> from scipy import sparse
>>> connected = adata.uns['paga']['connectivities'] > 0.1
>>> connected = (connected + connected.T) > 0
>>> idx_row, idx_col, dat = sparse.find(connected)
>>> idx = (idx_row >= idx_col)
>>> connected_celltypes = pd.DataFrame({ 'node1' : adata.obs['celltype'].cat.categories[idx_row[idx]],
                                         'node2' : adata.obs['celltype'].cat.categories[idx_col[idx]]})
>>> connected_celltypes
```

Check connected cell types

RNA velocity analysis with scVelo

```
>>> import scvelo as scv
>>> adata.raw = adata
>>> scv.pp.filter_and_normalize(adata,
                                min_shared_counts=10,
                                n_top_genes=3000)
>>> sc.pp.neighbors(adata, use_rep='X_pca')
>>> scv.pp.moments(adata, n_neighbors = None)
>>> scv.tl.velocity(adata, mode='stochastic')
>>> scv.tl.velocity_graph(adata)

>>> scv.pl.velocity_embedding_stream(adata, basis="umap", color="celltype", frameon=False)
>>> scv.pl.velocity_embedding_grid(adata, basis="umap", color="celltype", frameon=False,
arrow_size=2, arrow_length=2)
>>> scv.pl.velocity_embedding (adata, basis="umap", color="celltype", frameon=False,
arrow_size=2, arrow_length=2)
```



Fate probability estimation with CellRank 2

```
>>> import cellrank as cr

>>> pk = cr.kernels.PseudotimeKernel(adata, time_key="dpt_pseudotime").compute_transition_matrix()
>>> ck = cr.kernels.ConnectivityKernel(adata).compute_transition_matrix()
>>> vk = cr.kernels.VelocityKernel(adata).compute_transition_matrix()

>>> combined_kernel = 0.5 * vk + 0.3 * pk + 0.2 * ck
```

Generate hybrid kernels summarizing velocity, transcriptomic similarity (connectivity) and pseudotime

```
>>> g = cr.estimators.GPCCA(combined_kernel)
>>> g.fit(n_states=15, cluster_key="celltype")
>>> g.predict_terminal_states(method="top_n", n_states=10)
>>> g.plot_macrostates(which="terminal")
```

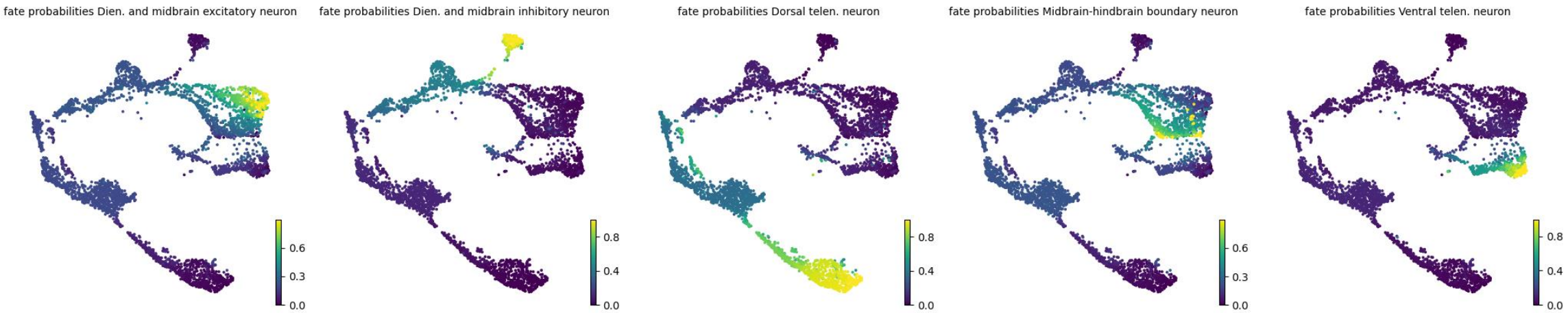
Predict terminal states

```
>>> g = cr.estimators.GPCCA(combined_kernel)
>>> neuron_types = [ x for x in adata.obs['celltype'].cat.categories if x.endswith('neuron') and
'early' not in x ]
>>> terminal_states =
[adata[adata.obs['celltype']==x,:].obs['dpt_pseudotime'].sort_values(ascending=False)[:30].index
for x in neuron_types]
>>> terminal_states = dict(zip(neuron_types, terminal_states))
>>> g.set_terminal_states(terminal_states)
>>> g.plot_macrostates(which="terminal")
```

Manually assign terminal states (each neuron type with the highest diffusion pseudotime)

Fate probability estimation with CellRank 2 (2)

```
>>> g.compute_fate_probabilities()
>>> g.plot_fate_probabilities(legend_loc="right", basis='X_umap', same_plot=False)
```



Questions?