How can frogs grow its tail back?

```
!pip install scanpy anndata scvi-tools leidenalg bbknn harmonypy magic-impute tabulate openpyxl igraph --quiet
import os
import re
import scanpy as sc
import anndata as ad
import numpy as np
import pandas as pd
from scipy.io import mmread
from scipy import sparse
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score, rand_score, silhouette_score
from sklearn.metrics import calinski_harabasz_score, davies_bouldin_score
from scipy.stats import hypergeom
import warnings
warnings.filterwarnings("ignore")
```

```
  Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.4/42.4 kB 3.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 647.5/647.5 kB 28.9 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 99.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 172.3/172.3 kB 15.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 628.9/628.9 kB 44.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.7/2.7 MB 111.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.7/5.7 MB 65.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.2/58.2 kB 4.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 50.1/50.1 kB 3.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 827.9/827.9 kB 54.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 104.4/104.4 kB 8.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 259.4/259.4 kB 23.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 983.2/983.2 kB 60.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 276.4/276.4 kB 27.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 76.7/76.7 kB 7.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 41.9/41.9 kB 3.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 756.0/756.0 kB 25.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.8/8.8 MB 115.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 35.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 183.0/183.0 kB 18.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 831.6/831.6 kB 62.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 80.0/80.0 kB 8.3 MB/s eta 0:00:00
  Building wheel for docrep (setup.py) ... done
  Building wheel for annoy (setup.py) ... done
```

Data accession

Load data from the link directly or load it through Courseworks .zip file.

```
!wget https://ftp.ebi.ac.uk/biostudies/fire/E-MTAB-/716/E-MTAB-7716/Files/arrayExpressUpload.zip -O /content/frogtai
```

```
--2025-11-11 01:30:38--  https://ftp.ebi.ac.uk/biostudies/fire/E-MTAB-/716/E-MTAB-7716/Files/arrayExpressUpload.zip
Resolving ftp.ebi.ac.uk (ftp.ebi.ac.uk)... 193.62.193.165
Connecting to ftp.ebi.ac.uk (ftp.ebi.ac.uk)|193.62.193.165|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 94419121 (90M) [application/zip]
Saving to: '/content/frogtail.zip'

/content/frogtail.z 100%[===================>]  90.04M   175MB/s    in 0.5s

2025-11-11 01:30:39 (175 MB/s) - '/content/frogtail.zip' saved [94419121/94419121]
```

```
import zipfile
import os

zip_file = "/content/frogtail.zip" # replace path to your zip file's path
extract_dir = "/content/drive/MyDrive/TA/Frogtail_files/" # where zip file is going to be extracted to

os.makedirs(extract_dir, exist_ok=True) # create the extraction directory if it doesn't exist

with zipfile.ZipFile(zip_file, 'r') as zip_ref: # unzip
```

```
        zip_ref.extractall(extract_dir)

print(f"Files extracted to {extract_dir}")
```

```
Files extracted to /content/drive/MyDrive/TA/Frogtail_files/
```

```
zip_file = extract_dir+"ArrayExpressV2.zip" # file is a nested zip, dezip again

with zipfile.ZipFile(zip_file, 'r') as zip_ref: # unzip
    zip_ref.extractall(extract_dir)

print(f"Files extracted to {extract_dir}")
```

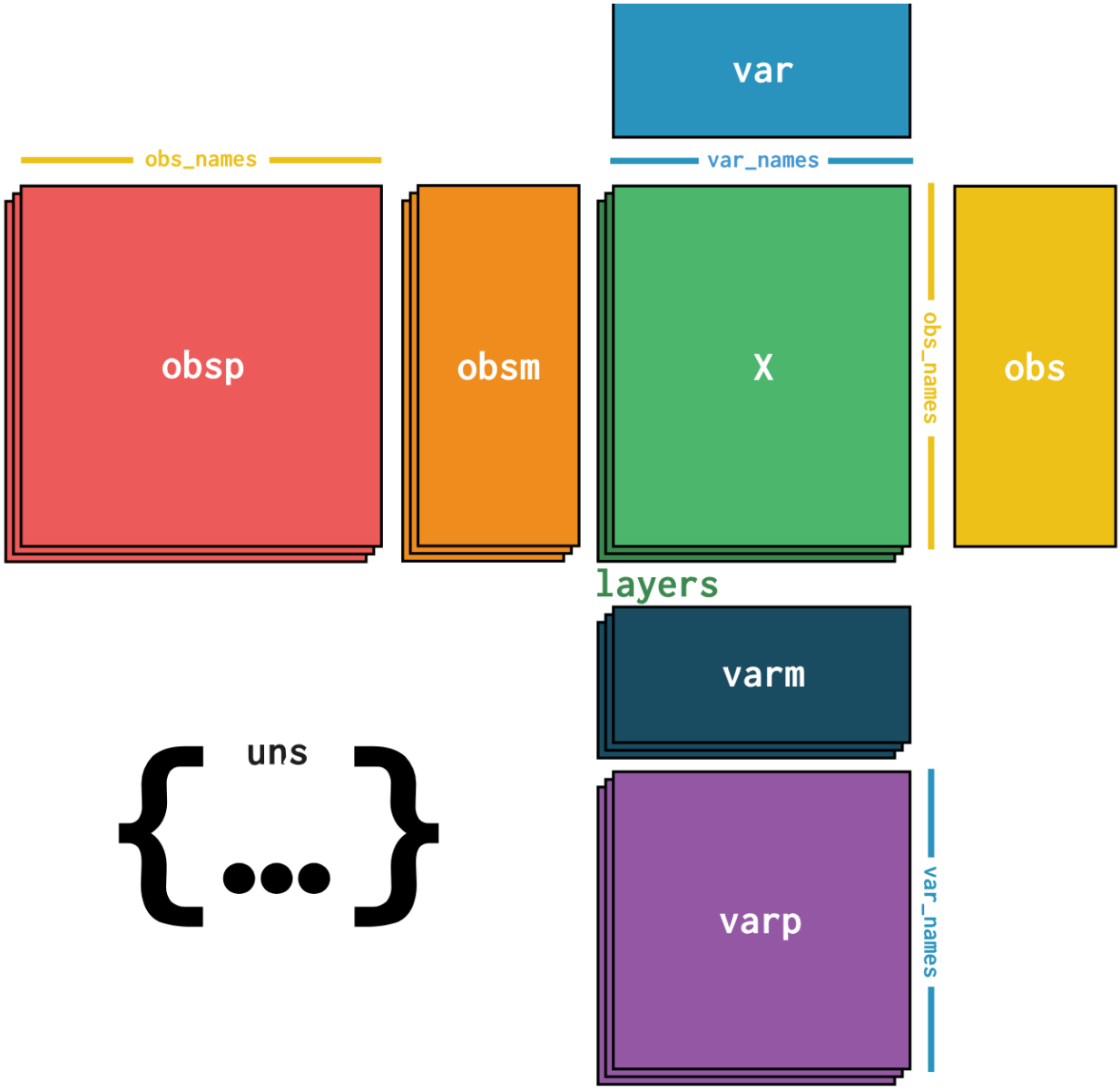
```
Files extracted to /content/drive/MyDrive/TA/Frogtail_files/
```

```
TABLE_S3_PATH = "/content/drive/MyDrive/aav9996_tables3.xlsx"
```

Make an anndata matrix using scanpy or anndata

Usually single cell data is either directly stored in highdimensional files with .h5 extensions or in compressed count matrices with several tsv/csv/json logging the covariates information. Here is an example of how to compress and download this data into an anndata object that facilitates downstream single cell analysis.

- AnnData object for single cell data: Documentation
- Scanpy for single cell analysis: Documentation




```
X = mmread(extract_dir+'ArrayExpress/countsMatrix.mtx') # compressed sparse matrix of gene expression
X = X.tocsr() # convert to sparse matrix
genes = pd.read_csv(extract_dir+'ArrayExpress/genes.csv', sep=' ', header=None) # col names of genes
cells = pd.read_csv(extract_dir+'ArrayExpress/cells.csv', sep=' ', header=None) # row names of cells
labels = pd.read_csv(extract_dir+'ArrayExpress/labels.csv')
meta = pd.read_csv(extract_dir+'ArrayExpress/meta.csv')


adata = ad.AnnData(X.T)
```

```
cells.columns = ['barcode_cells']
meta_aug = pd.merge(meta, labels, left_on='sample', right_on='Sample', how='left')
```

```
adata.var_names = genes[0]
adata.obs = pd.merge(cells, meta_aug, left_on='barcode_cells', right_on='cell')
```

```
adata
```

```
AnnData object with n_obs × n_vars = 13199 × 31535
    obs: 'barcode_cells', 'cell', 'sample', 'DevelopmentalStage', 'DaysPostAmputation', 'cluster', 'X', 'Y',
'CellCyclePhase', 'Sample', 'Lane', 'Condition', 'batch'
```

```
adata.var
```

Xelaev18000001m.g
Xelaev18000003m.g
Xelaev18000004m.g
Xelaev18000005m.g
tm6sf2.1
...
loc101731491.S
loc101731436.S
Xetrov90022661m.S
loc398467.S
Xetrov90026938m.S

31535 rows × 0 columns

```
adata.X = adata.X.tocsr()
```

```
adata.write_h5ad(extract_dir+'cleaned_processed_frogtail.h5ad')
```

Load data

```
import anndata as ad
extract_dir = "/content/drive/MyDrive/TA"
adata = ad.read_h5ad('/content/drive/MyDrive/TA/Frogtail_files/cleaned_processed_frogtail.h5ad') # ad.read_h5ad()
```

```
adata.X.todense()
```

```
matrix([[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]])
```

```
adata.obs
```

	barcode_cells	cell	sample	DevelopmentalStage	DaysPostAmputation	cluster	
0	AAACCTGAGCTAGTTC.1	AAACCTGAGCTAGTTC.1	SIGAB5	st40	3	Erythrocyte 4	-6.3954
1	AAACCTGGTGGGTCAA.1	AAACCTGGTGGGTCAA.1	SIGAB5	st40	3	Myeloid 1	-2.4282
2	AAACCTGGTTTGTTGG.1	AAACCTGGTTTGTTGG.1	SIGAB5	st40	3	Beta ionocyte	-1.3980
3	AAACGGGGTCGGCATC.1	AAACGGGGTCGGCATC.1	SIGAB5	st40	3	Erythrocyte 4	-5.8637
4	AAACGGGTCCTACAGA.1	AAACGGGTCCTACAGA.1	SIGAB5	st40	3	Goblet cell	2.1064
...	...	...	...	...	...	...	
13194	TTCTTAGAGTACCGGA.1	TTCTTAGAGTACCGGA.1	SIGAB10	st40	3	Erythrocyte 4	-5.8795
13195	TTGACTTAGAGTAAGG.1	TTGACTTAGAGTAAGG.1	SIGAB10	st40	3	Goblet cell	1.2521
13196	TTGCGTCTCAAGAAGT.1	TTGCGTCTCAAGAAGT.1	SIGAB10	st40	3	Goblet cell	0.6933
13197	TTGTAGGCAGTACACT.1	TTGTAGGCAGTACACT.1	SIGAB10	st40	3	Erythrocyte 1	-4.0876
13198	TTTGCGCAGCGTGAAC.1.1	TTTGCGCAGCGTGAAC.1.1	SIGAB10	st40	3	Goblet cell	0.8035

13199 rows × 13 columns

```
adata
```

```
AnnData object with n_obs × n_vars = 13199 × 31535
    obs: 'barcode_cells', 'cell', 'sample', 'DevelopmentalStage', 'DaysPostAmputation', 'cluster', 'X', 'Y',
'CellCyclePhase', 'Sample', 'Lane', 'Condition', 'batch'
```

```
adata.obs
```

	barcode_cells	cell	sample	DevelopmentalStage	DaysPostAmputation	cluster	
0	AAACCTGAGCTAGTTC.1	AAACCTGAGCTAGTTC.1	SIGAB5	st40	3	Erythrocyte 4	-6.3954
1	AAACCTGGTGGGTCAA.1	AAACCTGGTGGGTCAA.1	SIGAB5	st40	3	Myeloid 1	-2.4282
2	AAACCTGGTTTGTTGG.1	AAACCTGGTTTGTTGG.1	SIGAB5	st40	3	Beta ionocyte	-1.3980
3	AAACGGGGTCGGCATC.1	AAACGGGGTCGGCATC.1	SIGAB5	st40	3	Erythrocyte 4	-5.8637
4	AAACGGGTCCTACAGA.1	AAACGGGTCCTACAGA.1	SIGAB5	st40	3	Goblet cell	2.1064
...	...	...	...	...	...	...	
13194	TTCTTAGAGTACCGGA.1	TTCTTAGAGTACCGGA.1	SIGAB10	st40	3	Erythrocyte 4	-5.8795
13195	TTGACTTAGAGTAAGG.1	TTGACTTAGAGTAAGG.1	SIGAB10	st40	3	Goblet cell	1.2521
13196	TTGCGTCTCAAGAAGT.1	TTGCGTCTCAAGAAGT.1	SIGAB10	st40	3	Goblet cell	0.6933
13197	TTGTAGGCAGTACACT.1	TTGTAGGCAGTACACT.1	SIGAB10	st40	3	Erythrocyte 1	-4.0876
13198	TTTGCGCAGCGTGAAC.1.1	TTTGCGCAGCGTGAAC.1.1	SIGAB10	st40	3	Goblet cell	0.8035

13199 rows × 13 columns

Processing data

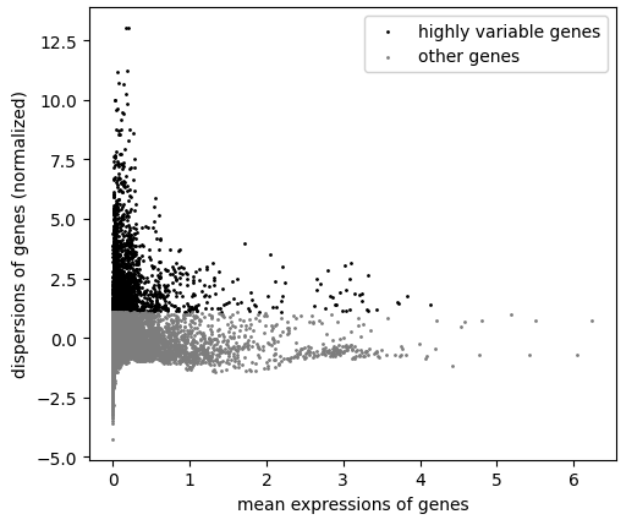
```
adata.layers["counts"] = adata.X.copy()
```

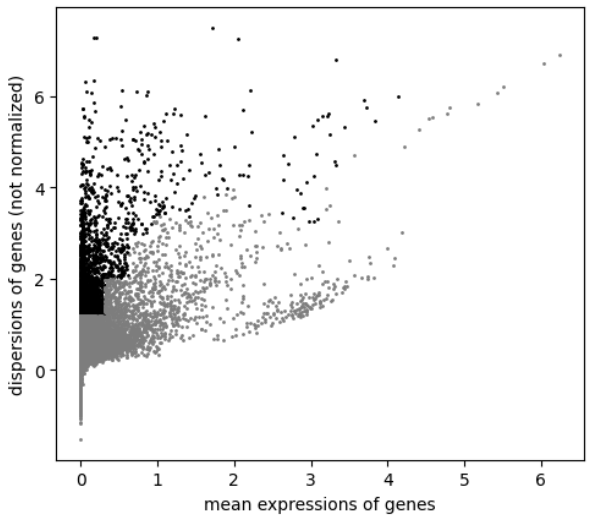
```
sc.pp.normalize_total(adata)
# Logarithmize the data
sc.pp.log1p(adata)
adata.raw = adata
```

```
sc.pp.filter_genes(adata, min_cells=3)
sc.pp.filter_cells(adata, min_genes=200)
sc.pp.highly_variable_genes(adata, n_top_genes=2300)
```

```
hvg = adata.var["highly_variable"].to_numpy()
adata.obsm["X_log2_hvg"] = adata[:, hvg].X
```

```
sc.pl.highly_variable_genes(adata)
```






```
from sklearn.cluster import KMeans
from sklearn.metrics import (
    adjusted_rand_score,
    rand_score,
    silhouette_score,
    calinski_harabasz_score,
    davies_bouldin_score,
)

print(adata)

# PCA on HVGs only
hvg_mask = adata.var["highly_variable"].values
adata_hvg = adata[:, hvg_mask].copy()

sc.pp.scale(adata_hvg, zero_center=True, max_value=10)
sc.tl.pca(adata_hvg, n_comps=50, svd_solver="arpack")

adata.obsm["X_pca"] = adata_hvg.obsm["X_pca"].copy()
```

```
AnnData object with n_obs × n_vars = 13199 × 26166
    obs: 'barcode_cells', 'cell', 'sample', 'DevelopmentalStage', 'DaysPostAmputation', 'cluster', 'X', 'Y', 'CellCyc
    var: 'n_cells', 'highly_variable', 'means', 'dispersions', 'dispersions_norm'
    uns: 'log1p', 'hvg'
    obsm: 'X_log2_hvg'
    layers: 'counts'
```

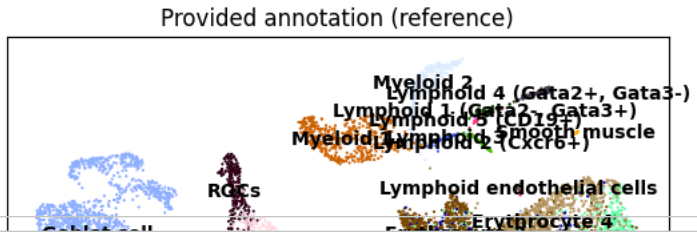
```
# Neighbors + UMAP
sc.pp.neighbors(adata, n_neighbors=15, n_pcs=30)
sc.tl.umap(adata)
```

```
#sc.tl.louvain(adata, key_added="louvain")
```

```
sc.tl.leiden(adata, key_added="leiden")
```

```
# Leiden (graph-based clustering)
sc.tl.leiden(adata, key_added="leiden")

# KMeans on PCA space
if "cluster" in adata.obs:
    n_true = int(adata.obs["cluster"].nunique())
else:
    n_true = 10

X_pca = adata.obsm["X_pca"]
kmeans = KMeans(n_clusters=n_true, random_state=0, n_init=10)
adata.obs["kmeans"] = kmeans.fit_predict(X_pca).astype(str)

# UMAP visualizations
sc.pl.umap(
    adata,
    color=["cluster"],
    title="Provided annotation (reference)",
    legend_loc="on data",
    wspace=0.4,
)
sc.pl.umap(
    adata,
    color=["leiden"],
    title="Leiden clusters",
    legend_loc="on data",
    wspace=0.4,
)
sc.pl.umap(
    adata,
    color=["kmeans"],
    title="KMeans clusters (PCA space)",
    legend_loc="on data",
    wspace=0.4,
)

# Metrics vs reference labels
if "cluster" in adata.obs:
    y_true = adata.obs["cluster"].astype("category")

    methods = []
    if "leiden" in adata.obs:
        methods.append("leiden")
    if "kmeans" in adata.obs:
        methods.append("kmeans")

    rows = []
    for key in methods:
        y_pred = adata.obs[key].astype("category")
        labs = y_pred.cat.codes.values

        if y_pred.nunique() > 1:
            ari = adjusted_rand_score(y_true, y_pred)
            ri = rand_score(y_true, y_pred)
            sil = silhouette_score(X_pca, labs)
            ch = calinski_harabasz_score(X_pca, labs)
            db = davies_bouldin_score(X_pca, labs)
        else:
            ari = ri = sil = ch = db = np.nan

        rows.append(
            dict(
                method=key,
                n_clusters=y_pred.nunique(),
```

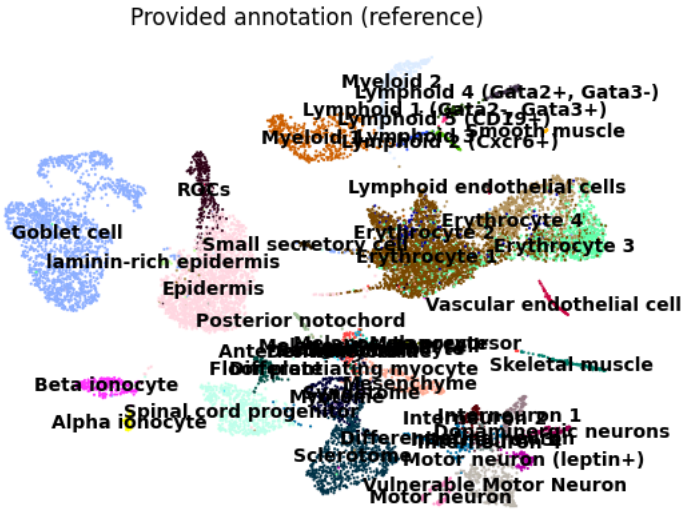
```
                    ARI=ari,
                    RandIndex=ri,
                    Silhouette_PCA=sil,
                    CalinskiHarabasz_PCA=ch,
                    DaviesBouldin_PCA=db,
                )
            )

        display(pd.DataFrame(rows))
    else:
        print("No 'cluster' column found in adata.obs to use as reference labels.")
```



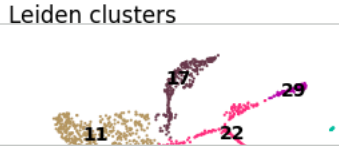

```
import os
import pandas as pd
import numpy as np
import scanpy as sc
from sklearn.metrics import (
    adjusted_rand_score,
    rand_score,
    silhouette_score,
    calinski_harabasz_score,
    davies_bouldin_score,
)


fig_dir = "./figures"
os.makedirs(fig_dir, exist_ok=True)


plot_keys = {
    "cluster": "Provided annotation (reference)",
    "leiden": "Leiden clusters",
    "kmeans": "KMeans clusters (PCA space)",
}

for key, title in plot_keys.items():
    if key in adata.obs:
        sc.pl.umap(
            adata,
            color=key,
            title=title,
            legend_loc="on data",
            frameon=False,
            wspace=0.4,
            show=True,
            save=f"_{key}.png",
        )

print(f"UMAP figures saved under {os.path.abspath('./figures')} "
      "(Scanpy may put them in ./figures/).")

#Metrics table vs ground truth labels

if "cluster" in adata.obs:
    if "X_pca" not in adata.obsm:
        raise ValueError("X_pca not found in adata.obsm. Run PCA before this cell.")

    y_true = adata.obs["cluster"].astype("category")
    X_pca = adata.obsm["X_pca"]

    results = []
    for key in ["leiden", "kmeans"]:
        if key not in adata.obs:
            continue

        y_pred = adata.obs[key].astype("category")
        n_clust = y_pred.nunique()
        labels_numeric = y_pred.cat.codes.values

        if n_clust > 1:
            ari = adjusted_rand_score(y_true, y_pred)
            ri = rand_score(y_true, y_pred)
            sil = silhouette_score(X_pca, labels_numeric)
            ch = calinski_harabasz_score(X_pca, labels_numeric)
            db = davies_bouldin_score(X_pca, labels_numeric)
        else:
            ari = ri = sil = ch = db = np.nan

        results.append(
```

```
                {
                    "method": key,
                    "n_clusters": n_clust,
                    "ARI_vs_ref": ari,
                    "RandIndex_vs_ref": ri,
                    "Silhouette_on_PCA": sil,
                    "CalinskiHarabasz_on_PCA": ch,
                    "DaviesBouldin_on_PCA": db,
                }
            )

        metrics_df = pd.DataFrame(results).set_index("method")
        display(
            metrics_df.style.format(
                {
                    "ARI_vs_ref": "{:.3f}",
                    "RandIndex_vs_ref": "{:.3f}",
                    "Silhouette_on_PCA": "{:.3f}",
                    "CalinskiHarabasz_on_PCA": "{:.1f}",
                    "DaviesBouldin_on_PCA": "{:.3f}",
                }
            )
        )
    else:
        print("No 'cluster' column in adata.obs; cannot compute metrics vs reference.")
```

```
WARNING: saving figure to file figures/umap_cluster.png
```




```
WARNING: saving figure to file figures/umap_leiden.png
```



Marker Selection

```
adata.obs['cluster'] = adata.obs['cluster'].astype('category')
print("Available clusters:\n", adata.obs['cluster'].cat.categories)

# 1. Select ROC cluster label explicitly
if "ROCs" not in adata.obs['cluster'].cat.categories:
    raise ValueError("Expected 'ROCs' in cluster labels; please check adata.obs['cluster'].")

roc_label = "ROCs"
print(f"Using ROC cluster label: {roc_label}")

# 2. Marker selection: Wilcoxon (ROC vs rest)
sc.tl.rank_genes_groups(
    adata,
    groupby='cluster',
    groups=[roc_label],
    reference='rest',
    method='wilcoxon',
    use_raw=True,
    key_added='markers_wilcoxon'
)

# 3. Marker selection: Logistic Regression on ALL clusters
sc.tl.rank_genes_groups(
    adata,
    groupby='cluster',
    method='logreg',
    use_raw=True,
    key_added='markers_logreg_all'
)

# convert rank_genes_groups output -> DataFrame
def get_marker_df(adata, key, group, n_top=200):
    rg = adata.uns[key]
    names = rg['names'][group][:n_top]
    scores = rg['scores'][group][:n_top]

    if 'pvals_adj' in rg:
        pvals = rg['pvals_adj'][group][:n_top]
    else:
        pvals = rg['pvals'][group][:n_top] if 'pvals' in rg else np.full(len(names), np.nan)

    if 'logfoldchanges' in rg:
        logfc = rg['logfoldchanges'][group][:n_top]
    else:
        logfc = np.full(len(names), np.nan)
```

```
        return pd.DataFrame({
            'gene': names,
            'score': scores,
            'logFC': logfc,
            'pval_adj': pvals
        })

    # ROC markers from both methods
    roc_wilcoxon = get_marker_df(adata, 'markers_wilcoxon', roc_label, n_top=200)
    roc_logreg   = get_marker_df(adata, 'markers_logreg_all', roc_label, n_top=200)

    # Overlap between methods
    overlap = pd.merge(
        roc_wilcoxon,
        roc_logreg,
        on='gene',
        suffixes=('_wilcoxon', '_logreg')
    )

    print("\nTop 10 ROC markers by Wilcoxon:")
    display(roc_wilcoxon.head(10))

    print("\nTop 10 ROC markers by Logistic Regression:")
    display(roc_logreg.head(10))

    print("\nOverlap between Wilcoxon + LogReg (top 20 by Wilcoxon score):")
    display(overlap.sort_values('score_wilcoxon', ascending=False).head(20))

    # 5. Compare with Supplementary Table 3 gene set
    TABLE_S3_PATH = "/content/drive/MyDrive/aav9996_tables3.xlsx"

    try:
        s3 = pd.read_excel(TABLE_S3_PATH, sheet_name=0)
    except Exception as e:
        print("\nCould not read Supplementary Table 3 file:", e)
        s3 = None

    import re
    import pandas as pd

    #  Canonicalization helper
    def canon(g):
        """
        Normalize gene names:
        - strip whitespace
        - remove trailing .L / .S (Xenopus isoforms)
        - drop non-alphanumeric chars
        - uppercase
        """
        g = str(g).strip()
        g = re.sub(r'\.(L|S)$', '', g)      # remove only suffix .L or .S
        g = re.sub(r'[^A-Za-z0-9]', '', g)  # remove anything weird
        return g.upper()

    #  Table 3 gene list

    # (a) Try from Excel if loaded earlier
    table3_genes = []

    if s3 is not None:
        # assume single gene column or first column is gene names
        gene_col = s3.columns[0]
        table3_genes = (
            s3[gene_col]
            .dropna()
            .astype(str)
            .tolist()
        )

    # (b) Manual override using the list you pasted (this guarantees correctness)
    manual_table3 = [
        "wnt5a",
        "loc100488523",
        "loc100492954",
        "egfl6",
        "lpar3",
        "epo6"
```

```
    "cpa6",
    "sp9",
    "fgf9",
    "fbn2",
    "tinagl1",
    "fgf7",
    "vwde",
    "lef1",
    "lamb2",
    "fgfr4",
    "fgf7",
    "rspo2",
    "cpa6",
    "loc100498358",
    "krt",
    "loc100486832",
    "jag1",
    "Xelaev18043128m",
    "dlx2",
    "tp73",
    "gdf6",
    "Xelaev18046800m",
    "nid2",
    "igfbp2",
    "ism2",
    "dlx2",
    "loc100493805",
    "frem2",
    "Xelaev18044182m",
    "bmp5",
    "tspear",
    "sema3f",
    "robo4",
    "jchain",
    "rspo2",
    "adamts18",
    "pltp",
    "Xelaev18034799m",
    "lamb1",
    "lamb1",
    "unc5b",
    "mmp28",
    "fgf10",
    "pzp",
    "galnt15",
]

# if manual list is present, use it as ground truth
if manual_table3:
    table3_genes = manual_table3

# canonical set
table3_genes_c = {canon(g) for g in table3_genes if g}

print(f"Loaded {len(table3_genes)} raw Table 3 entries, {len(table3_genes_c)} unique canonical IDs.")

# Canonicalize ROC markers
roc_wilcoxon_c = roc_wilcoxon.copy()
roc_logreg_c   = roc_logreg.copy()
overlap_c      = overlap.copy()

roc_wilcoxon_c["gene_c"] = roc_wilcoxon_c["gene"].map(canon)
roc_logreg_c["gene_c"]   = roc_logreg_c["gene"].map(canon)
overlap_c["gene_c"]      = overlap_c["gene"].map(canon)

# Compute overlaps
wilcoxon_in_s3_c = roc_wilcoxon_c[roc_wilcoxon_c["gene_c"].isin(table3_genes_c)]
logreg_in_s3_c   = roc_logreg_c[roc_logreg_c["gene_c"].isin(table3_genes_c)]
overlap_in_both_c = overlap_c[overlap_c["gene_c"].isin(table3_genes_c)]

print("\nCanonical overlap counts:")
print(f" Wilcoxon ROC markers overlapping Table 3: {len(wilcoxon_in_s3_c)}")
print(f" LogReg ROC markers overlapping Table 3: {len(logreg_in_s3_c)}")
print(f" High-confidence (in both methods + Table 3): {len(overlap_in_both_c)}")

print("\nExamples: Wilcoxon ROC markers overlapping Table 3 (canonical):")
display(wilcoxon_in_s3_c.head(20))
```

```
print("\nExamples: LogReg ROC markers overlapping Table 3 (canonical):")
display(logreg_in_s3_c.head(20))

print("\nExamples: high-confidence ROC markers (both methods + Table 3, canonical):")
display(overlap_in_both_c.head(20))
```

```
Available clusters:
 Index(['Alpha ionocyte', 'Anterior notochord', 'Beta ionocyte', 'Dermomyotome',
       'Differentiating myocyte', 'Differentiating neuron',
       'Dopaminergic neurons', 'Epidermis', 'Erythrocyte 1', 'Erythrocyte 2',
       'Erythrocyte 3', 'Erythrocyte 4', 'Floor plate', 'Goblet cell',
       'Interneuron 1', 'Interneuron 2', 'Interneuron 3', 'Interneuron 4',
       'Lymphoid 1 (Gata2−, Gata3+)', 'Lymphoid 2 (Cxcr6+)', 'Lymphoid 3',
       'Lymphoid 4 (Gata2+, Gata3−)', 'Lymphoid 5 (CD19+)',
       'Lymphoid endothelial cells', 'Melanocyte', 'Melanocyte precursor',
       'Melanocyte stem cell', 'Mesenchyme', 'Motor neuron',
       'Motor neuron (leptin+)', 'Myeloid 1', 'Myeloid 2', 'Myotome',
       'Oligodendrocyte', 'Posterior notochord', 'ROCs', 'Satellite cell',
       'Sclerotome', 'Skeletal muscle', 'Small secretory cell',
       'Smooth muscle', 'Spinal cord progenitor', 'Syndetome',
       'Vascular endothelial cell', 'Vulnerable Motor Neuron',
       'laminin−rich epidermis'],
      dtype='object')
Using ROC cluster label: ROCs

Top 10 ROC markers by Wilcoxon:
```

	gene	score	logFC	pval_adj
0	col14a1.S	25.357527	5.884367	2.340832e-137
1	apoc1.like.L	25.222475	8.242413	3.580654e-136
2	loc100486548.L	25.060778	5.087452	1.400259e-134
3	mdk.L	24.477623	4.950447	2.016198e-128
4	id3.S	23.945599	4.214293	6.476695e-123
5	lama5.L	23.924074	5.031434	9.042865e-123
6	fras1.L	23.778688	5.202516	2.500209e-121
7	col14a1.L	23.365553	4.591443	3.775154e-117
8	id3.L	23.067509	3.929649	3.439342e-114
9	bambi.L	22.982174	4.741649	2.216280e-113

```
Top 10 ROC markers by Logistic Regression:
```

	gene	score	logFC	pval_adj
0	apoc1.like.L	0.486242	NaN	NaN
1	pltp.S	0.418675	NaN	NaN
2	fn1.S	0.329667	NaN	NaN
3	id3.L	0.296707	NaN	NaN
4	Xetrov90029035m.L	0.287145	NaN	NaN
5	nid2.L	0.278743	NaN	NaN
6	frem2.1.L	0.266963	NaN	NaN
7	loc100486548.L	0.259576	NaN	NaN
8	fstl1.L	0.247599	NaN	NaN
9	krt5.7.S	0.247056	NaN	NaN

```
import numpy as np
import pandas as pd
import scanpy as sc
from sklearn.cluster import KMeans
from sklearn.metrics import (
    adjusted_rand_score,
    rand_score,
    silhouette_score,
    calinski_harabasz_score,
    davies_bouldin_score,
)

def run_clustering(
    adata,
    prefix="base",
    ref_key="cluster",
    use_rep=None,
    n_neighbors=15,
    n_pcs=30,
    n_comps_pca=50,
):
```

```
    """
    Run neighbors/UMAP + Leiden + KMeans.

    Works on:
      - current adata + HVG-based PCA (if use_rep is None)
      - an existing embedding in adata.obsm[use_rep] (e.g. 'X_scvi', 'X_pca_magic').

    Stores labels in:
      - f'{prefix}_leiden'
      - f'{prefix}_kmeans'

    Returns:
      - metrics_df: ARI / Rand / Silhouette / CH / DB vs ref_key (if present).
    """

    # 1. Choose representation
    if use_rep is not None:
        if use_rep not in adata.obsm:
            raise ValueError(f"use_rep='{use_rep}' not found in adata.obsm")
        X_for_metrics = adata.obsm[use_rep]
        sc.pp.neighbors(adata, n_neighbors=n_neighbors, use_rep=use_rep)
    else:
        # build PCA on HVGs safely (no dense full matrix)
        if "highly_variable" not in adata.var:
            raise ValueError(
                "No 'highly_variable' in adata.var. "
                "Run sc.pp.highly_variable_genes before run_clustering."
            )

        hvg_mask = adata.var["highly_variable"].values
        adata_hvg = adata[:, hvg_mask].copy()

        sc.pp.scale(adata_hvg, zero_center=True, max_value=10)
        sc.tl.pca(adata_hvg, n_comps=n_comps_pca, svd_solver="arpack")

        # store PCA in main object
        adata.obsm["X_pca"] = adata_hvg.obsm["X_pca"].copy()
        X_for_metrics = adata.obsm["X_pca"]

        sc.pp.neighbors(adata, n_neighbors=n_neighbors, n_pcs=n_pcs)

    # UMAP
    sc.tl.umap(adata)

    # Leiden clustering
    leiden_key = f"{prefix}_leiden"
    sc.tl.leiden(adata, key_added=leiden_key)

    # ----------------------------
    # 4. KMeans on chosen representation
    # ----------------------------
    if ref_key in adata.obs:
        n_true = int(adata.obs[ref_key].nunique())
    else:
        n_true = 20  # fallback

    km_key = f"{prefix}_kmeans"
    km = KMeans(n_clusters=n_true, random_state=0, n_init=10)
    adata.obs[km_key] = km.fit_predict(X_for_metrics).astype(str)

    # ----------------------------
    # 5. Metrics vs reference
    # ----------------------------
    results = []
    methods = [leiden_key, km_key]

    if ref_key in adata.obs:
        y_true = adata.obs[ref_key].astype("category")

        for key in methods:
            y_pred = adata.obs[key].astype("category")
            n_clust = y_pred.nunique()

            if n_clust > 1:
                labels_num = y_pred.cat.codes.to_numpy()
                ari = adjusted_rand_score(y_true, y_pred)
                ri = rand_score(y_true, y_pred)
                sil = silhouette_score(X_for_metrics, labels_num)
```

```
                sil = silhouette_score(X_for_metrics, labels_num)
                ch = calinski_harabasz_score(X_for_metrics, labels_num)
                db = davies_bouldin_score(X_for_metrics, labels_num)
            else:
                ari = ri = sil = ch = db = np.nan

            results.append(
                dict(
                    method=key,
                    n_clusters=n_clust,
                    ARI_vs_ref=ari,
                    RandIndex_vs_ref=ri,
                    Silhouette=sil,
                    CalinskiHarabasz=ch,
                    DaviesBouldin=db,
                )
            )

    metrics_df = pd.DataFrame(results)
    return metrics_df
```

11	egfl6 S	22.220493	4.526450	5.719964e-106	0.113467	NaN	NaN	EGFL6

```
import re
import numpy as np
import pandas as pd
import scanpy as sc

TABLE3_GENES = [
    "wnt5a","loc100488523","loc100492954","egfl6","lpar3","cpa6","sp9","fgf9","fbn2",
    "tinagl1","fgf7","vwde","lef1","lamb2","fgfr4","fgf7","rspo2","cpa6","loc100498358",
    "krt","loc100486832","jag1","Xelaev18043128m","dlx2","tp73","gdf6","Xelaev18046800m",
    "nid2","igfbp2","ism2","dlx2","loc100493805","frem2","Xelaev18044182m","bmp5","tspear",
    "sema3f","robo4","jchain","rspo2","adamts18","pltp","Xelaev18034799m","lamb1","lamb1",
    "unc5b","mmp28","fgf10","pzp","galnt15",
]

def canon(g):
    g = str(g).strip()
    g = re.sub(r'\.(L|S)$', '', g)
    g = re.sub(r'[^A-Za-z0-9]', '', g)
    return g.upper()

TABLE3_CANON = {canon(g) for g in TABLE3_GENES if g}

def run_roc_markers(
    adata,
    cluster_key="cluster",
    roc_label="ROCs",
    key_prefix="base",
    n_top=200,
    expression_source="raw",   # 'raw' or 'X'
):
    # --- checks ---
    if cluster_key not in adata.obs:
        raise ValueError(f"{cluster_key} not found in adata.obs")

    if roc_label not in adata.obs[cluster_key].unique():
        raise ValueError(f"{roc_label} not found in {cluster_key} categories")

    if expression_source not in {"raw", "X"}:
        raise ValueError("expression_source must be 'raw' or 'X'")

    if expression_source == "raw":
        if adata.raw is None:
            raise ValueError(
                "expression_source='raw' but adata.raw is None. "
                "Set adata.raw before calling or use expression_source='X'."
            )
        use_raw_flag = True
    else:
        use_raw_flag = False

    # Wilcoxon: ROC vs rest
    w_key = f"{key_prefix}_markers_wilcoxon"
    sc.tl.rank_genes_groups(
        adata,
        groupby=cluster_key,
```

```
        groups=[roc_label],
        reference="rest",
        method="wilcoxon",
        use_raw=use_raw_flag,
        key_added=w_key,
    )

    # LogReg: all clusters
    l_key = f"{key_prefix}_markers_logreg_all"
    sc.tl.rank_genes_groups(
        adata,
        groupby=cluster_key,
        method="logreg",
        use_raw=use_raw_flag,
        key_added=l_key,
    )

    #  helper to extract into DF
    def _get_df(key, group):
        rg = adata.uns[key]
        names = rg["names"][group][:n_top]
        scores = rg["scores"][group][:n_top]

        if "pvals_adj" in rg:
            pvals = rg["pvals_adj"][group][:n_top]
        elif "pvals" in rg:
            pvals = rg["pvals"][group][:n_top]
        else:
            pvals = np.full(len(names), np.nan)

        if "logfoldchanges" in rg:
            logfc = rg["logfoldchanges"][group][:n_top]
        else:
            logfc = np.full(len(names), np.nan)

        return pd.DataFrame(
            {"gene": names, "score": scores, "logFC": logfc, "pval_adj": pvals}
        )

    roc_w = _get_df(w_key, roc_label)
    roc_l = _get_df(l_key, roc_label)

    #overlap Wilcoxon & LogReg
    overlap = pd.merge(
        roc_w, roc_l, on="gene", suffixes=("_wilcoxon", "_logreg")
    )

    # Table 3 overlaps (canonical)
    roc_w_c = roc_w.copy()
    roc_l_c = roc_l.copy()
    overlap_c = overlap.copy()

    roc_w_c["gene_c"] = roc_w_c["gene"].map(canon)
    roc_l_c["gene_c"] = roc_l_c["gene"].map(canon)
    overlap_c["gene_c"] = overlap_c["gene"].map(canon)

    w_in_t3 = roc_w_c[roc_w_c["gene_c"].isin(TABLE3_CANON)]
    l_in_t3 = roc_l_c[roc_l_c["gene_c"].isin(TABLE3_CANON)]
    both_in_t3 = overlap_c[overlap_c["gene_c"].isin(TABLE3_CANON)]

    print(
        f"{key_prefix} [{expression_source}]: "
        f"WilcoxonnTable3={len(w_in_t3)}, "
        f"LogRegnTable3={len(l_in_t3)}, "
        f"BothMethodsnTable3={len(both_in_t3)}"
    )

    return roc_w, roc_l, overlap, w_in_t3, l_in_t3, both_in_t3
```

Data Denoising

Magic Denoising Method

```
import magic

# Copy to avoid touching original
adata_magic = adata.copy()

# Run MAGIC on existing log-normalized X
magic_op = magic.MAGIC()
adata_magic.X = magic_op.fit_transform(adata_magic.X)

# Clustering on MAGIC-smoothed expression
magic_metrics = run_clustering(
    adata_magic,
    prefix="magic",
    ref_key="cluster",   # compare to true annotations
    use_rep=None         # neighbors/PCA on MAGIC X
)
print("MAGIC clustering metrics:")
display(magic_metrics)

# ROC markers using MAGIC-denoised expression
magic_roc_w, magic_roc_l, magic_overlap, magic_w_t3, magic_l_t3, magic_both_t3 = run_roc_markers(
    adata_magic,
    cluster_key="cluster",     # ROC cluster from annotation
    roc_label="ROCs",
    key_prefix="magic",
    expression_source="X",     # <- use denoised X
)

print("\nTop 5 MAGIC ROC markers (Wilcoxon):")
display(magic_roc_w.head(5))

print("\nTop 5 MAGIC ROC markers (LogReg):")
display(magic_roc_l.head(5))

display(magic_both_t3)
```

```
Start coding or generate with AI.
```

ScVi Denoising Method

```
import scvi
import torch

torch.set_float32_matmul_precision("high")

adata_scvi = adata.copy()

scvi.model.SCVI.setup_anndata(
    adata_scvi,
    layer="counts",
    batch_key="batch",
)

scvi_model = scvi.model.SCVI(adata_scvi, n_latent=20)
scvi_model.train()

adata_scvi.obsm["X_scvi"] = scvi_model.get_latent_representation()

# 2) Clustering on scVI latent (Leiden + KMeans via your helper)
scvi_metrics = run_clustering(
    adata_scvi,
    prefix="scvi",
    ref_key="cluster",
    use_rep="X_scvi",
```

```
)
print("scVI clustering metrics:")
display(scvi_metrics)

# 3) Use scVI normalized expression as adata_scvi.X for marker calling
scvi_norm = scvi_model.get_normalized_expression()
adata_scvi.X = scvi_norm.to_numpy()

# 4) ROC markers on scVI-denoised expression
scvi_roc_w, scvi_roc_l, scvi_overlap, scvi_w_t3, scvi_l_t3, scvi_both_t3 = run_roc_markers(
    adata_scvi,
    cluster_key="cluster",
    roc_label="ROCs",
    key_prefix="scvi",
    expression_source="X",
)

print("\nTop 5 scVI ROC markers (Wilcoxon):")
display(scvi_roc_w.head(5))

print("\nTop 5 scVI ROC markers (LogReg):")
display(scvi_roc_l.head(5))
```

```
INFO: GPU available: True (cuda), used: True
INFO:lightning.pytorch.utilities.rank_zero:GPU available: True (cuda), used: True
INFO: TPU available: False, using: 0 TPU cores
INFO:lightning.pytorch.utilities.rank_zero:TPU available: False, using: 0 TPU cores
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
Epoch 400/400: 100%                                                    400/400 [10:11<00:00,  1.53s/it, v_num=1, train_loss=6.29e+3]
INFO: `Trainer.fit` stopped: `max_epochs=400` reached.
INFO:lightning.pytorch.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=400` reached.
scVI clustering metrics:
```

	method	n_clusters	ARI_vs_ref	RandIndex_vs_ref	Silhouette	CalinskiHarabasz	DaviesBouldin
0	scvi_leiden	29	0.592881	0.934191	0.141092	567.823242	1.789734
1	scvi_kmeans	46	0.333059	0.915269	0.133442	518.081909	1.875293

```
scvi [X]: WilcoxonTable3=50, LogRegTable3=12, BothMethodsTable3=10

Top 5 scVI ROC markers (Wilcoxon):
```

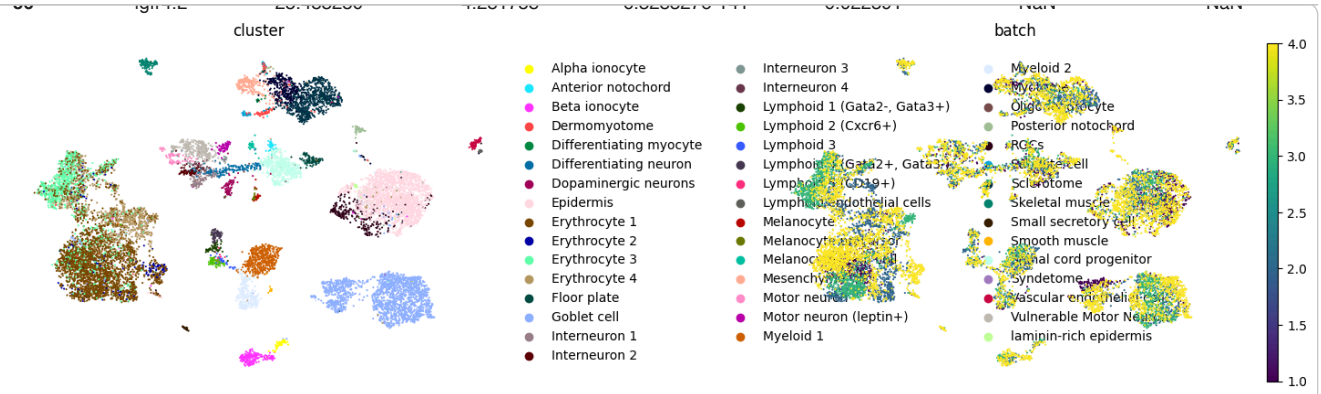
	gene	score	logFC	pval_adj
0	Xetrov90029035m.L	26.877703	6.560441	1.047511e-154
1	lamb2.L	26.801929	4.723192	3.796655e-154
2	egfl6.L	26.779732	4.917590	3.796655e-154
3	tinagl1.S	26.778168	5.712176	3.796655e-154
4	cpa6.L	26.728617	5.689699	1.145561e-153

```
Top 5 scVI ROC markers (LogReg):
```

	gene	score	logFC	pval_adj
0	apoc1.like.L	5.901590	NaN	NaN
2	Xelaev18022166m.g	1.736227	NaN	NaN
4	krt12.L	0.820547	NaN	NaN

	gene	pval_adj_wilcoxon	score_logreg	logFC_logreg	pval_adj_logreg	
0	lpar3.L	1.753888e-153	0.033600	NaN	NaN	
2	egfl6.L	5.942070e-153	0.023415	NaN	NaN	
3	tinagl1.S	1.862615e-151	0.033296	NaN	NaN	
4	cpa6.L	1.983330e-151	0.034435	NaN	NaN	
8	cpa6.L	2.360872e-151	0.034362	NaN	NaN	
9	egfl6.L	4.979359e-151	0.038122	NaN	NaN	
13	tspan4.L	8.036773e-151	0.048447	NaN	NaN	
26	krt5.7.S	4.378570e-148	0.081427	NaN	NaN	
32	unc5b.L	5.524220e-146	0.031944	NaN	NaN	
50	nid2.L	2.834985e-142	0.071239	NaN	NaN	
56	fgfr4.L	6.328327e-141	0.022391	NaN	NaN	

```
sc.pp.neighbors(adata_scvi, use_rep="X_scvi", n_neighbors=15)
sc.tl.umap(adata_scvi)

sc.pl.umap(
    adata_scvi,
    color=["cluster", "batch"],
    wspace=0.4,
    frameon=False,
)
```




```
import re
import numpy as np
import pandas as pd
import scanpy as sc

TABLE3_GENES = [
    "wnt5a","loc100488523","loc100492954","egfl6","lpar3","cpa6","sp9","fgf9","fbn2",
    "tinagl1","fgf7","vwde","lef1","lamb2","fgfr4","fgf7","rspo2","cpa6","loc100498358",
    "krt","loc100486832","jag1","Xelaev18043128m","dlx2","tp73","gdf6","Xelaev18046800m",
    "nid2","igfbp2","ism2","dlx2","loc100493805","frem2","Xelaev18044182m","bmp5","tspear",
    "sema3f","robo4","jchain","rspo2","adamts18","pltp","Xelaev18034799m","lamb1","lamb1",
    "unc5b","mmp28","fgf10","pzp","galnt15",
]

def canon(g):
    g = str(g).strip()
    g = re.sub(r'\.(L|S)$', '', g)
    g = re.sub(r'[^A-Za-z0-9]', '', g)
    return g.upper()

TABLE3_CANON = {canon(g) for g in TABLE3_GENES if g}

def run_roc_markers(
    adata,
    cluster_key="cluster",
    roc_label="ROCs",
    key_prefix="base",
    n_top=200,
    expression_source="raw",
):
    # checks
    if cluster_key not in adata.obs:
        raise ValueError(f"{cluster_key} not found in adata.obs")

    # enforce categorical for safety
    if not pd.api.types.is_categorical_dtype(adata.obs[cluster_key]):
        adata.obs[cluster_key] = adata.obs[cluster_key].astype("category")

    cats = list(adata.obs[cluster_key].cat.categories)
    if roc_label not in cats:
        raise ValueError(
            f"{roc_label} not found in {cluster_key} categories.\n"
            f"Available: {cats}"
        )

    if expression_source not in {"raw", "X"}:
        raise ValueError("expression_source must be 'raw' or 'X'")
    use_raw_flag = (expression_source == "raw")

    #  Wilcoxon: ROC vs rest
    sc.tl.rank_genes_groups(
        adata,
        groupby=cluster_key,
        groups=[roc_label],
        reference="rest",
```

```
        method="wilcoxon",
        use_raw=use_raw_flag,
        key_added=f"{key_prefix}_markers_wilcoxon",
    )

    #  LogReg: all clusters
    sc.tl.rank_genes_groups(
        adata,
        groupby=cluster_key,
        method="logreg",
        use_raw=use_raw_flag,
        key_added=f"{key_prefix}_markers_logreg_all",
    )

    def _get_df(key, group):
        rg = adata.uns[key]
        names = rg["names"][group][:n_top]
        scores = rg["scores"][group][:n_top]

        if "pvals_adj" in rg:
            pvals = rg["pvals_adj"][group][:n_top]
        elif "pvals" in rg:
            pvals = rg["pvals"][group][:n_top]
        else:
            pvals = np.full(len(names), np.nan)

        if "logfoldchanges" in rg:
            logfc = rg["logfoldchanges"][group][:n_top]
        else:
            logfc = np.full(len(names), np.nan)

        return pd.DataFrame(
            {"gene": names, "score": scores, "logFC": logfc, "pval_adj": pvals}
        )

    w_key = f"{key_prefix}_markers_wilcoxon"
    l_key = f"{key_prefix}_markers_logreg_all"

    roc_w = _get_df(w_key, roc_label)
    roc_l = _get_df(l_key, roc_label)

    overlap = pd.merge(roc_w, roc_l, on="gene", suffixes=("_wilcoxon", "_logreg"))

    # overlap with Table 3
    roc_w_c = roc_w.copy(); roc_w_c["gene_c"] = roc_w_c["gene"].map(canon)
    roc_l_c = roc_l.copy(); roc_l_c["gene_c"] = roc_l_c["gene"].map(canon)
    overlap_c = overlap.copy(); overlap_c["gene_c"] = overlap_c["gene"].map(canon)

    w_in_t3 = roc_w_c[roc_w_c["gene_c"].isin(TABLE3_CANON)]
    l_in_t3 = roc_l_c[roc_l_c["gene_c"].isin(TABLE3_CANON)]
    both_in_t3 = overlap_c[overlap_c["gene_c"].isin(TABLE3_CANON)]

    print(
        f"{key_prefix} [{expression_source}]: "
        f"WilcoxonnTable3={len(w_in_t3)}, "
        f"LogRegnTable3={len(l_in_t3)}, "
        f"BothMethodsnTable3={len(both_in_t3)}"
    )

    return roc_w, roc_l, overlap, w_in_t3, l_in_t3, both_in_t3
```

```
import scanpy.external as sce

adata_harmony = adata.copy()

#check:
if "batch" not in adata_harmony.obs.columns:
    raise ValueError("adata.obs['batch'] missing — required for Harmony.")
if "highly_variable" not in adata_harmony.var:
    raise ValueError("adata.var['highly_variable'] missing — run highly_variable_genes first.")

adata_harmony.obs["batch"] = adata_harmony.obs["batch"].astype("category")

hvg_mask = adata_harmony.var["highly_variable"].values
adata_hvg = adata_harmony[:, hvg_mask].copy()
```

```
sc.pp.scale(adata_hvg, zero_center=True, max_value=10)
sc.tl.pca(adata_hvg, n_comps=50, svd_solver="arpack")

adata_harmony.obsm["X_pca"] = adata_hvg.obsm["X_pca"].copy()

# Harmony integration:
sce.pp.harmony_integrate(
    adata_harmony,
    key="batch",
    basis="X_pca",
    adjusted_basis="X_pca_harmony"
)
```

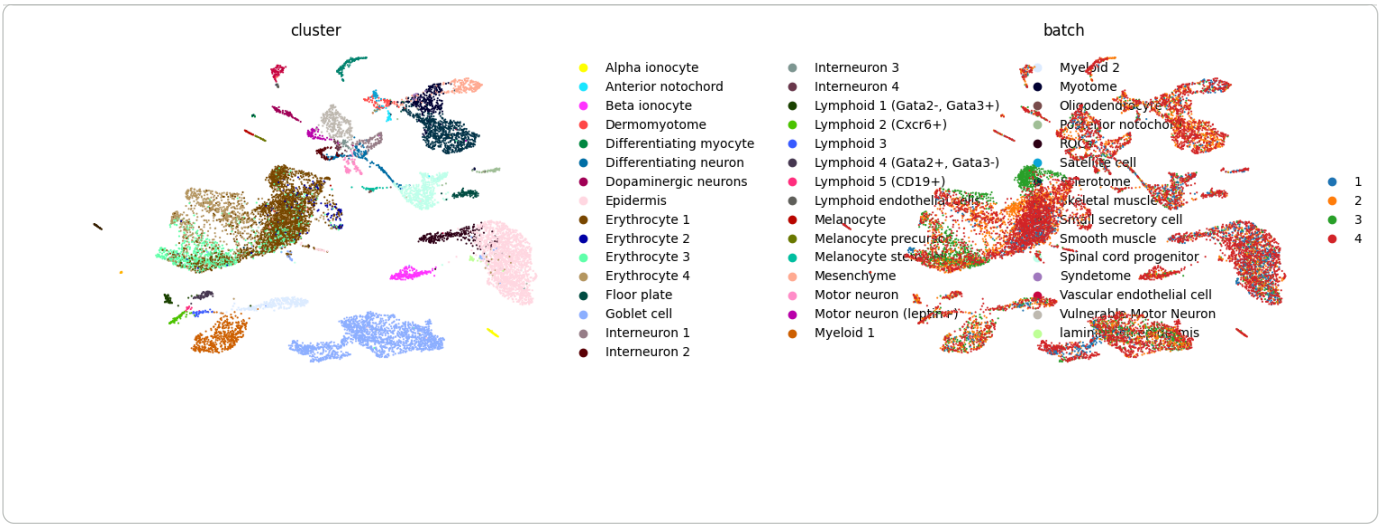
```
2025-11-11 01:53:57,073 - harmonypy - INFO - Computing initial centroids with sklearn.KMeans...
INFO:harmonypy:Computing initial centroids with sklearn.KMeans...
2025-11-11 01:53:59,789 - harmonypy - INFO - sklearn.KMeans initialization complete.
INFO:harmonypy:sklearn.KMeans initialization complete.
2025-11-11 01:53:59,839 - harmonypy - INFO - Iteration 1 of 10
INFO:harmonypy:Iteration 1 of 10
2025-11-11 01:54:02,310 - harmonypy - INFO - Iteration 2 of 10
INFO:harmonypy:Iteration 2 of 10
2025-11-11 01:54:04,781 - harmonypy - INFO - Iteration 3 of 10
INFO:harmonypy:Iteration 3 of 10
2025-11-11 01:54:07,900 - harmonypy - INFO - Converged after 3 iterations
INFO:harmonypy:Converged after 3 iterations
```

```
# clustering on Harmony PCs with your helper
harmony_metrics = run_clustering(
    adata_harmony,
    prefix="harmony",
    ref_key="cluster",
    use_rep="X_pca_harmony",
)
display(harmony_metrics)

# markers still on raw or baseline expression
harmony_roc_w, harmony_roc_l, harmony_overlap, \
harmony_w_t3, harmony_l_t3, harmony_both_t3 = run_roc_markers(
    adata_harmony,
    cluster_key="cluster",
    roc_label="ROCs",
    key_prefix="harmony",
    expression_source="raw",
)
```

	method	n_clusters	ARI_vs_ref	RandIndex_vs_ref	Silhouette	CalinskiHarabasz	DaviesBouldin
0	harmony_leiden	38	0.483911	0.927514	0.255459	1645.723018	1.153158
1	harmony_kmeans	46	0.535782	0.916502	0.325212	2113.136242	1.124882

```
harmony [raw]: WilcoxonnTable3=19, LogRegnTable3=28, BothMethodsnTable3=16
```

```
# neighbors on Harmony PCs
sc.pp.neighbors(adata_harmony, use_rep="X_pca_harmony", n_neighbors=15)
sc.tl.umap(adata_harmony)

sc.pl.umap(
    adata_harmony,
    color=["cluster", "batch"],
    wspace=0.4,
    frameon=False,
)
```



Batch Integration

```
import scanpy.external as sce

# BBKNN on PCA (batch integration)
adata_bbknn = adata.copy()

if "highly_variable" not in adata_bbknn.var:
    raise ValueError("Run sc.pp.highly_variable_genes on `adata` before this block.")

# PCA on HVGs only
hvg_mask = adata_bbknn.var["highly_variable"].values
adata_bbknn_hvg = adata_bbknn[:, hvg_mask].copy()
sc.pp.scale(adata_bbknn_hvg, zero_center=True, max_value=10)
sc.tl.pca(adata_bbknn_hvg, n_comps=50, svd_solver="arpack")
adata_bbknn.obsm["X_pca"] = adata_bbknn_hvg.obsm["X_pca"].copy()

# BBKNN neighbors by batch (uses X_pca)
sce.pp.bbknn(adata_bbknn, batch_key="batch")

# UMAP on BBKNN graph
sc.tl.umap(adata_bbknn)

# Leiden on BBKNN graph
sc.tl.leiden(adata_bbknn, key_added="bbknn_leiden")

# KMeans on PCA (for comparison)
if "cluster" in adata_bbknn.obs:
    n_true = int(adata_bbknn.obs["cluster"].nunique())
else:
    n_true = 20

X_pca_bb = adata_bbknn.obsm["X_pca"]
km = KMeans(n_clusters=n_true, random_state=0, n_init=10)
adata_bbknn.obs["bbknn_kmeans"] = km.fit_predict(X_pca_bb).astype(str)

# --- Metrics vs annotated clusters (using X_pca for both) ---
bb_rows = []
if "cluster" in adata_bbknn.obs:
    y_true = adata_bbknn.obs["cluster"].astype("category")

    for key in ["bbknn_leiden", "bbknn_kmeans"]:
        y_pred = adata_bbknn.obs[key].astype("category")
        n_clust = y_pred.nunique()

        if n_clust > 1:
            labs = y_pred.cat.codes.values
            ari = adjusted_rand_score(y_true, y_pred)
            ri = rand_score(y_true, y_pred)
            sil = silhouette_score(X_pca_bb, labs)
            ch = calinski_harabasz_score(X_pca_bb, labs)
            db = davies_bouldin_score(X_pca_bb, labs)
        else:
```

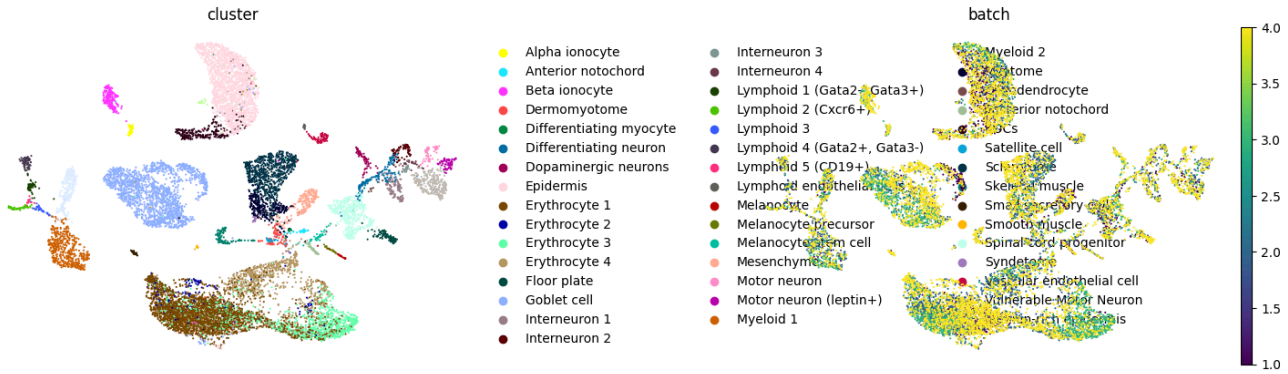
```
            ari = ri = sil = ch = db = np.nan

        bb_rows.append({
            "method": key,
            "n_clusters": n_clust,
            "ARI_vs_ref": ari,
            "RandIndex_vs_ref": ri,
            "Silhouette": sil,
            "CalinskiHarabasz": ch,
            "DaviesBouldin": db,
        })

bbknn_metrics = pd.DataFrame(bb_rows)
print("BBKNN clustering metrics:")
display(bbknn_metrics)

# ROC markers using raw expression
bbknn_roc_w, bbknn_roc_l, bbknn_overlap, bbknn_w_t3, bbknn_l_t3, bbknn_both_t3 = run_roc_markers(
    adata_bbknn,
    cluster_key="cluster",
    roc_label="ROCs",
    key_prefix="bbknn",
    expression_source="raw",
)
```

```
WARNING: consider updating your call to make use of `computation`
BBKNN clustering metrics:
```

	method	n_clusters	ARI_vs_ref	RandIndex_vs_ref	Silhouette	CalinskiHarabasz	DaviesBouldin
0	bbknn_leiden	34	0.511020	0.929677	0.245883	1584.144287	1.227045
1	bbknn_kmeans	46	0.526036	0.914686	0.349193	1972.115967	1.152038

```
bbknn [raw]: WilcoxonnTable3=19, LogRegnTable3=28, BothMethodsnTable3=16
```

```
sc.pl.umap(
    adata_bbknn,
    color=["cluster", "batch"],
    wspace=0.4,
    frameon=False,
)
```




```
base_roc_w, base_roc_l, base_overlap, base_w_t3, base_l_t3, base_both_t3= run_roc_markers(adata,         key_prefix=

def top_genes(df, n=50):
    return set(df["gene"][:n])

marker_sets = {
    "base":    top_genes(base_roc_w),
    "magic":   top_genes(magic_roc_w),
    "scvi":    top_genes(scvi_roc_w),
    "harmony": top_genes(harmony_roc_w),
    "bbknn":   top_genes(bbknn_roc_w),
}
```

```
# Overlap of each method with base
rows = []
for name, genes in marker_sets.items():
    if name == "base":
        continue
    rows.append({
        "pipeline": name,
        "overlap_with_base_top50": len(genes & marker_sets["base"]),
    })

markers_overlap_df = pd.DataFrame(rows)
display(markers_overlap_df)
```

```
base [raw]: WilcoxonnTable3=19, LogRegnTable3=28, BothMethodsnTable3=16
```

	pipeline	overlap_with_base_top50
0	magic	15
1	scvi	16
2	harmony	50
3	bbknn	50

```
def count_table3_overlap(df, n=50):
    return sum(canon(g) in TABLE3_CANON for g in df["gene"][:n])

rows = []
rows.append({"pipeline": "base",    "top50_in_Table3": count_table3_overlap(base_roc_w)})
rows.append({"pipeline": "magic",   "top50_in_Table3": count_table3_overlap(magic_roc_w)})
rows.append({"pipeline": "scvi",    "top50_in_Table3": count_table3_overlap(scvi_roc_w)})
rows.append({"pipeline": "harmony", "top50_in_Table3": count_table3_overlap(harmony_roc_w)})
rows.append({"pipeline": "bbknn",   "top50_in_Table3": count_table3_overlap(bbknn_roc_w)})

table3_comp = pd.DataFrame(rows)
display(table3_comp)
```

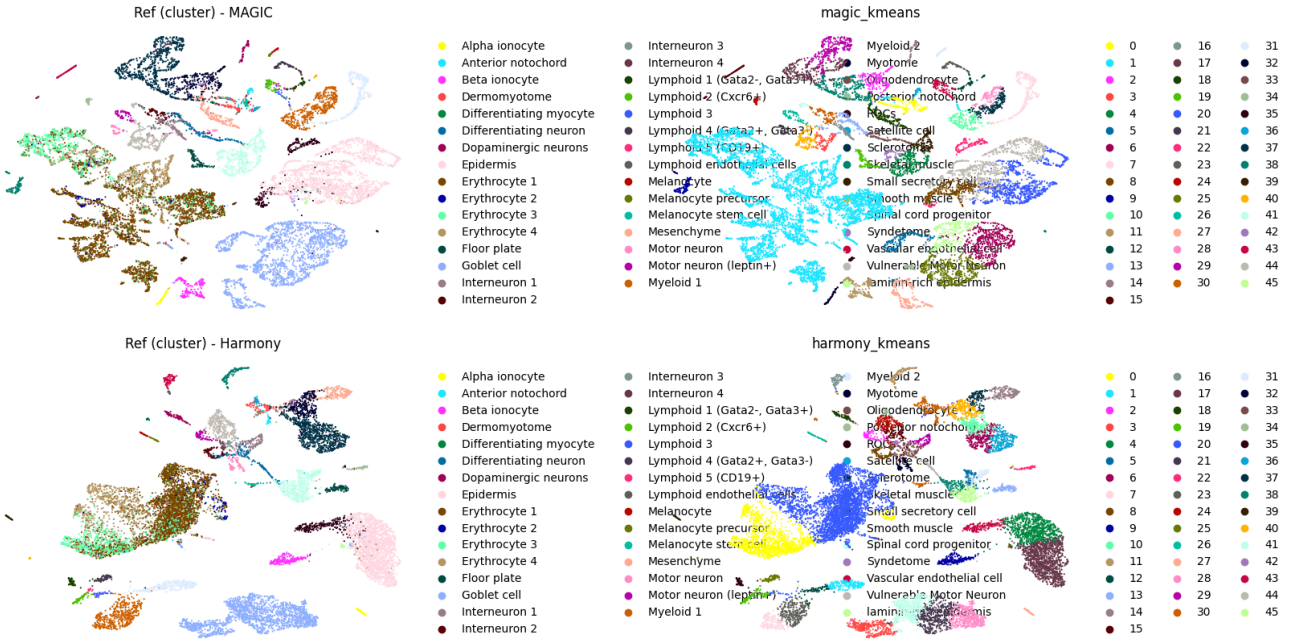
	pipeline	top50_in_Table3
0	base	5
1	magic	20
2	scvi	19
3	harmony	5
4	bbknn	5

```
all_metrics = []

# (1) Base
base_metrics = run_clustering(adata, prefix="base", ref_key="cluster", use_rep=None)
base_metrics["pipeline"] = "base"
all_metrics.append(base_metrics)

# (2) MAGIC
magic_metrics["pipeline"] = "magic"
all_metrics.append(magic_metrics)

# (3) scVI (uses latent)
scvi_metrics["pipeline"] = "scvi"
all_metrics.append(scvi_metrics)

# (4) Harmony
harmony_metrics["pipeline"] = "harmony"
all_metrics.append(harmony_metrics)

# (5) BBKNN
bbknn_metrics["pipeline"] = "bbknn"
all_metrics.append(bbknn_metrics)

metrics_all = pd.concat(all_metrics, ignore_index=True)

display(
```

```
    metrics_all[["pipeline", "method", "n_clusters",
                 "ARI_vs_ref", "Silhouette",
                 "CalinskiHarabasz", "DaviesBouldin"]]
    .sort_values(["ARI_vs_ref"], ascending=False)
)
```

	pipeline	method	n_clusters	ARI_vs_ref	Silhouette	CalinskiHarabasz	DaviesBouldin
4	scvi	scvi_leiden	29	0.592881	0.141092	567.823242	1.789734
7	harmony	harmony_kmeans	46	0.535782	0.325212	2113.136242	1.124882
9	bbknn	bbknn_kmeans	46	0.526036	0.349193	1972.115967	1.152038
1	base	base_kmeans	46	0.526036	0.349193	1972.115967	1.152038
8	bbknn	bbknn_leiden	34	0.511020	0.245883	1584.144287	1.227045
0	base	base_leiden	36	0.484483	0.241852	1532.947266	1.318344
6	harmony	harmony_leiden	38	0.483911	0.255459	1645.723018	1.153158
3	magic	magic_kmeans	46	0.478626	0.571742	3849.712646	0.750189
5	scvi	scvi_kmeans	46	0.333059	0.133442	518.081909	1.875293
2	magic	magic_leiden	58	0.284729	0.297470	2014.850708	1.156042

```
# 1) MAGIC: reference vs magic_kmeans
sc.pl.umap(
    adata_magic,
    color=["cluster", "magic_kmeans"],
    ncols=2,
    frameon=False,
    wspace=0.4,
    title=["Ref (cluster) – MAGIC", "magic_kmeans"],
)

# 2) Harmony: reference vs harmony_kmeans
sc.pl.umap(
    adata_harmony,
    color=["cluster", "harmony_kmeans"],
    ncols=2,
    frameon=False,
    wspace=0.4,
    title=["Ref (cluster) – Harmony", "harmony_kmeans"],
)
```




```
cluster_to_category = {
"Erythrocyte 4": "Red blood cells",
"Myeloid 1": "Immune system",
"Beta ionocyte": "Skin",
"Goblet cell": "Skin",
"Erythrocyte 1": "Red blood cells",
"Epidermis": "Skin",
"Sclerotome": "Somites",
"Myotome": "Somites",
"Spinal cord progenitor": "Neural",
"Myeloid 2": "Immune system",
"Mesenchyme": "Somites",
"ROCs": "Skin",
"Melanocyte": "Skin",
"Erythrocyte 3": "Red blood cells",
"Differentiating neuron": "Neural",
"Satellite cell": "Somites",
"Vascular endothelial cell": "Somites",
"Vulnerable Motor Neuron": "Neural",
"Alpha ionocyte": "Skin",
"Lymphoid 1 (Gata2−, Gata3+)": "Immune system",
"Lymphoid 2 (Cxcr6+)": "Immune system",
"Erythrocyte 2": "Red blood cells",
"Interneuron 1": "Neural",
"Skeletal muscle": "Somites",
"Lymphoid 4 (Gata2+, Gata3−)": "Immune system",
"Small secretory cell": "Skin",
"Floor plate": "Neural",
"Dermomyotome": "Somites",
"Interneuron 2": "Neural",
"Dopaminergic neurons": "Neural",
"Lymphoid 5 (CD19+)": "Immune system",
"Interneuron 4": "Neural",
"Melanocyte stem cell": "Skin",
"laminin−rich epidermis": "Skin",
"Motor neuron (leptin+)": "Neural",
"Motor neuron": "Neural",
"Lymphoid 3": "Immune system",
"Posterior notochord": "Somites",
"Melanocyte precursor": "Skin",
```

```
"Anterior notochord": "Somites",
"Interneuron 3": "Neural",
"Smooth muscle": "Somites",
"Syndetome": "Somites",
"Differentiating myocyte": "Somites",
"Oligodendrocyte": "Neural",
"Lymphoid endothelial cells": "Immune system"
}

adata.obs["category"] = adata.obs["cluster"].map(cluster_to_category).astype("category")
```

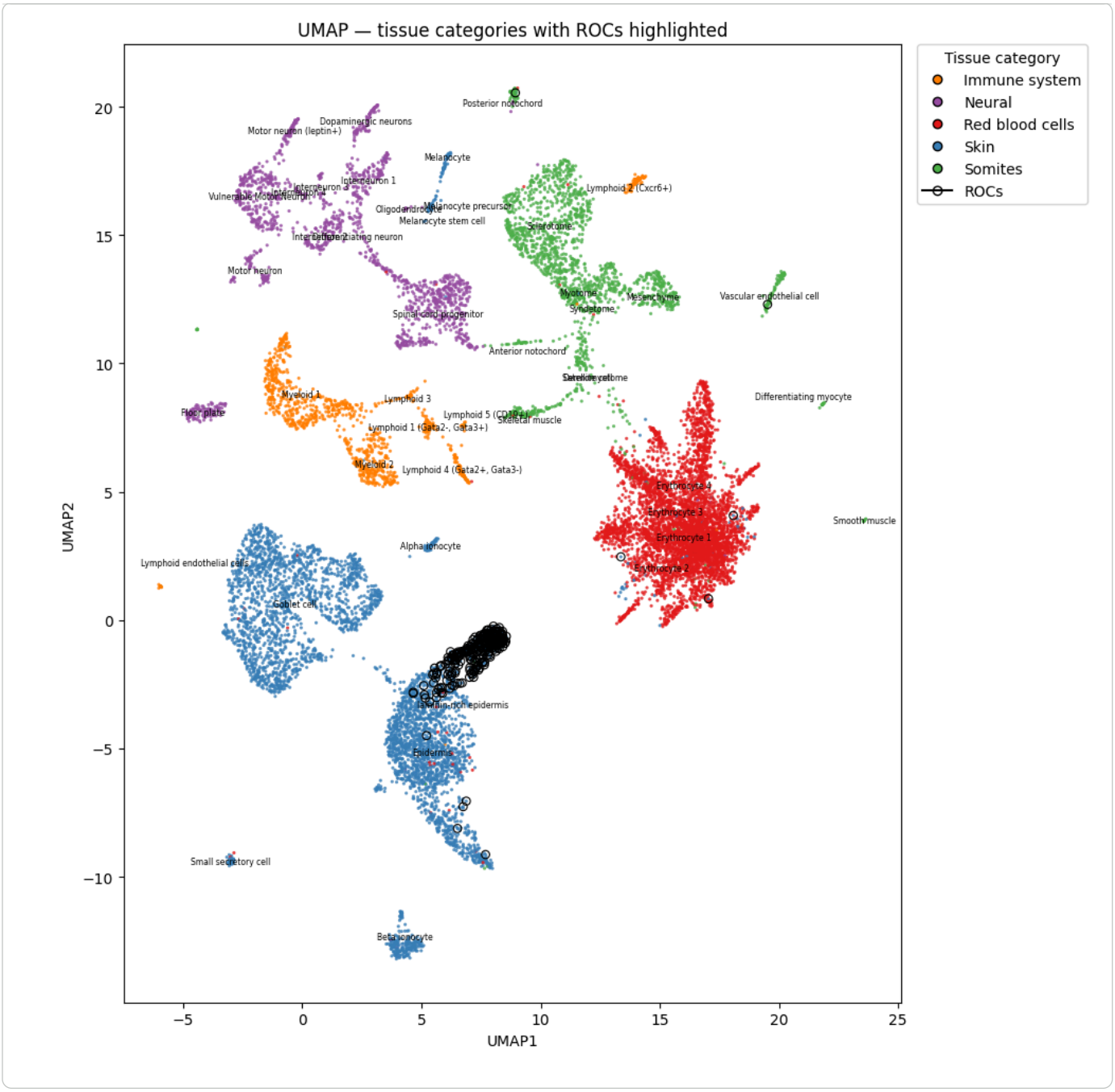
```
import anndata as ad
import numpy as np
import pandas as pd
import scanpy as sc
import scipy.sparse as sp
from scipy.io import mmread
import matplotlib.pyplot as plt

# 1. Load raw data
extract_dir = "/content/drive/MyDrive/TA/Frogtail_files/"

X = mmread(extract_dir + "ArrayExpress/countsMatrix.mtx").tocsr()
genes = pd.read_csv(extract_dir + "ArrayExpress/genes.csv", sep=" ", header=None)
cells = pd.read_csv(extract_dir + "ArrayExpress/cells.csv", sep=" ", header=None)
labels = pd.read_csv(extract_dir + "ArrayExpress/labels.csv")
meta = pd.read_csv(extract_dir + "ArrayExpress/meta.csv")

cells.columns = ["barcode_cells"]

adata_raw = ad.AnnData(X.T)
adata_raw.var_names = genes[0]

meta_aug = pd.merge(meta, labels, left_on="sample", right_on="Sample", how="left")
adata_raw.obs = pd.merge(cells, meta_aug, left_on="barcode_cells", right_on="cell")

print("Raw AnnData:", adata_raw)


adata = adata_raw.copy()
adata.layers["counts"] = adata.X.copy()

# e normalize
sc.pp.normalize_total(adata, target_sum=1e4)

# log2(x+1)
if sp.issparse(adata.X):
    Xn = adata.X.toarray()
else:
    Xn = adata.X
adata.X = np.log2(Xn + 1)

#HVGs via Fano + mean filters (paper-style)

Xlog = adata.X
means = Xlog.mean(axis=0)
vars_ = Xlog.var(axis=0, ddof=1)

# Fano factor
fano = np.divide(vars_, means, out=np.zeros_like(vars_), where=means > 0)

thr = np.percentile(fano, 65)
hvg = fano > thr

low, high = np.percentile(means[hvg], [5, 80])
keep = hvg & (means >= low) & (means <= high)

adata = adata[:, keep].copy()

nonzero_mask = np.array((adata.X > 0).sum(axis=1)).ravel() > 0
adata = adata[nonzero_mask, :].copy()

print("After paper-style HVG & filters:", adata)
```

```
sc.pp.neighbors(
    adata,
    n_neighbors=20,
    metric="cosine",
    method="umap",
)

sc.tl.umap(
    adata,
    min_dist=0.5,
)
```

```
Raw AnnData: AnnData object with n_obs × n_vars = 13199 × 31535
    obs: 'barcode_cells', 'cell', 'sample', 'DevelopmentalStage', 'DaysPostAmputation', 'cluster', 'X', 'Y', 'CellCyc
After paper-style HVG & filters: AnnData object with n_obs × n_vars = 13199 × 8277
    obs: 'barcode_cells', 'cell', 'sample', 'DevelopmentalStage', 'DaysPostAmputation', 'cluster', 'X', 'Y', 'CellCyc
    layers: 'counts'
```

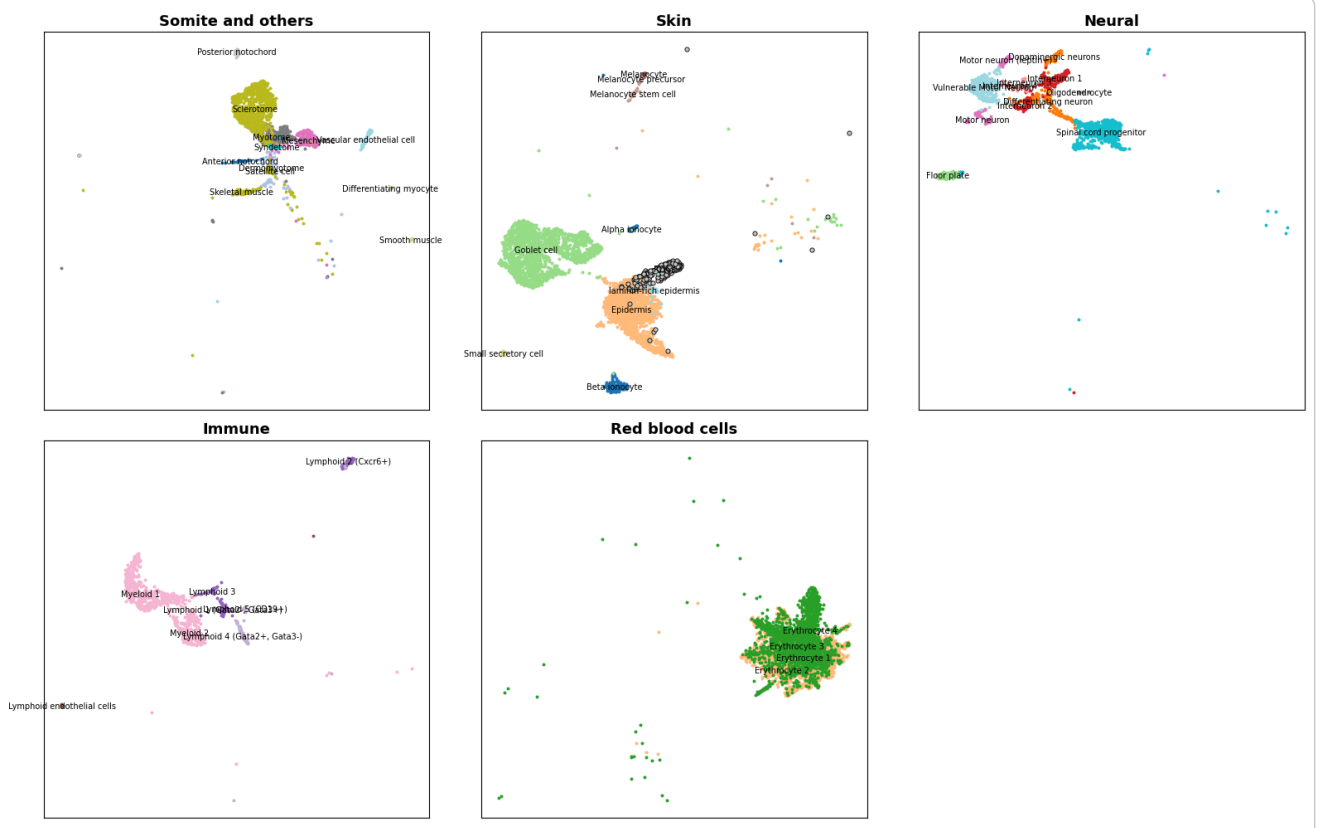
```
from matplotlib.lines import Line2D

if "X_umap" not in adata.obsm:
    raise ValueError("adata.obsm['X_umap'] not found. Run UMAP before this cell.")

if "cluster" not in adata.obs:
    raise ValueError("adata.obs['cluster'] not found.")

if "category" not in adata.obs:
    # if missing, build from cluster_to_category
    if "cluster_to_category" in globals():
        adata.obs["category"] = (
            adata.obs["cluster"].astype(str).map(cluster_to_category)
            .fillna("Other/Unknown")
            .astype("category")
        )
    else:
        raise ValueError("adata.obs['category'] not found and cluster_to_category not defined.")

clusters = adata.obs["cluster"].astype(str)
cats = adata.obs["category"].astype(str)

coords = adata.obsm["X_umap"]

cat_colors = {
    "Red blood cells": "#e41a1c",
    "Skin": "#377eb8",
    "Somites": "#4daf4a",
    "Neural": "#984ea3",
    "Immune system": "#ff7f00",
    "Other/Unknown": "#999999",
}

unique_cats = sorted(cats.unique())
for c in unique_cats:
    if c not in cat_colors:
        cat_colors[c] = "#999999"

# map each cell to category color
point_colors = cats.map(cat_colors).to_numpy()

plt.figure(figsize=(10, 10))
plt.scatter(
    coords[:, 0],
    coords[:, 1],
    c=point_colors,
    s=5,
    linewidths=0,
    alpha=0.8,
)

if "ROCs" in clusters.unique():
    roc_mask = clusters == "ROCs"
    plt.scatter(
        coords[roc_mask, 0],
```

```
            coords[roc_mask, 1],
            facecolors="none",
            edgecolors="black",
            s=30,
            linewidths=0.8,
            label="ROCs",
        )

    for cl in sorted(clusters.unique()):
        mask = clusters == cl
        if mask.sum() == 0:
            continue
        x = coords[mask, 0].mean()
        y = coords[mask, 1].mean()
        plt.text(
            x,
            y,
            cl,
            fontsize=5.5,
            ha="center",
            va="center",
            color="black",
        )


    #legend for tissue categories (+ ROCs)
    legend_elems = [
        Line2D(
            [0], [0],
            marker="o",
            color="none",
            markerfacecolor=cat_colors[c],
            markersize=6,
            label=c,
        )
        for c in unique_cats
    ]

    if "ROCs" in clusters.unique():
        legend_elems.append(
            Line2D(
                [0], [0],
                marker="o",
                color="black",
                markerfacecolor="none",
                markersize=6,
                label="ROCs",
            )
        )

    plt.legend(
        handles=legend_elems,
        bbox_to_anchor=(1.02, 1),
        loc="upper left",
        borderaxespad=0.0,
        title="Tissue category",
    )

    plt.xlabel("UMAP1")
    plt.ylabel("UMAP2")
    plt.title("UMAP — tissue categories with ROCs highlighted")
    plt.tight_layout()
    plt.show()
```




```
X = adata.obsm["X_umap"][:, 0]
Y = adata.obsm["X_umap"][:, 1]

clusters = adata.obs["cluster"].astype("category").cat.categories

cluster_colors = {}
if "cluster_colors" in adata.uns and len(adata.uns["cluster_colors"]) == len(clusters):
    for cl, c in zip(clusters, adata.uns["cluster_colors"]):
        cluster_colors[cl] = c
else:
    cmap = plt.get_cmap("tab20")
    n = len(clusters)
    for i, cl in enumerate(clusters):
        cluster_colors[cl] = cmap(i / max(1, n - 1))

panel_order = [
    ("Somites", "Somite and others"),
    ("Skin", "Skin"),
    ("Neural", "Neural"),
    ("Immune system", "Immune"),
    ("Red blood cells", "Red blood cells"),
]
```

```
fig, axes = plt.subplots(2, 3, figsize=(16, 10))
axes = axes.flatten()
for ax in axes[5:]:
    ax.axis("off")

for (cat, title), ax in zip(panel_order, axes):
    # subset: only cells in this category
    mask_cat = (adata.obs["category"] == cat).to_numpy()
    if not np.any(mask_cat):
        ax.axis("off")
        continue

    # clusters present in this category
    clusters_cat = (
        adata.obs.loc[mask_cat, "cluster"]
        .astype("category")
        .cat.remove_unused_categories()
        .cat.categories
    )

    # plot each cluster in this category separately
    for cl in clusters_cat:
        m = mask_cat & (adata.obs["cluster"] == cl).to_numpy()
        if not np.any(m):
            continue

        # highlight ROC
        if cl == "ROCs":
            ax.scatter(
                X[m],
                Y[m],
                s=14,
                c=cluster_colors.get(cl, "red"),
                edgecolors="black",
                linewidths=0.6,
            )
        else:
            ax.scatter(
                X[m],
                Y[m],
                s=8,
                c=cluster_colors.get(cl, "C0"),
                linewidths=0,
            )

    # text labels at median of each cluster (within this category only)
    for cl in clusters_cat:
        m = mask_cat & (adata.obs["cluster"] == cl).to_numpy()
        if not np.any(m):
            continue
        xm = np.median(X[m])
        ym = np.median(Y[m])
        ax.text(
            xm,
            ym,
            cl,
            fontsize=7,
            ha="center",
            va="center",
        )

    ax.set_title(title, fontsize=13, fontweight="bold")
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_frame_on(True)

plt.tight_layout()
plt.show()
```

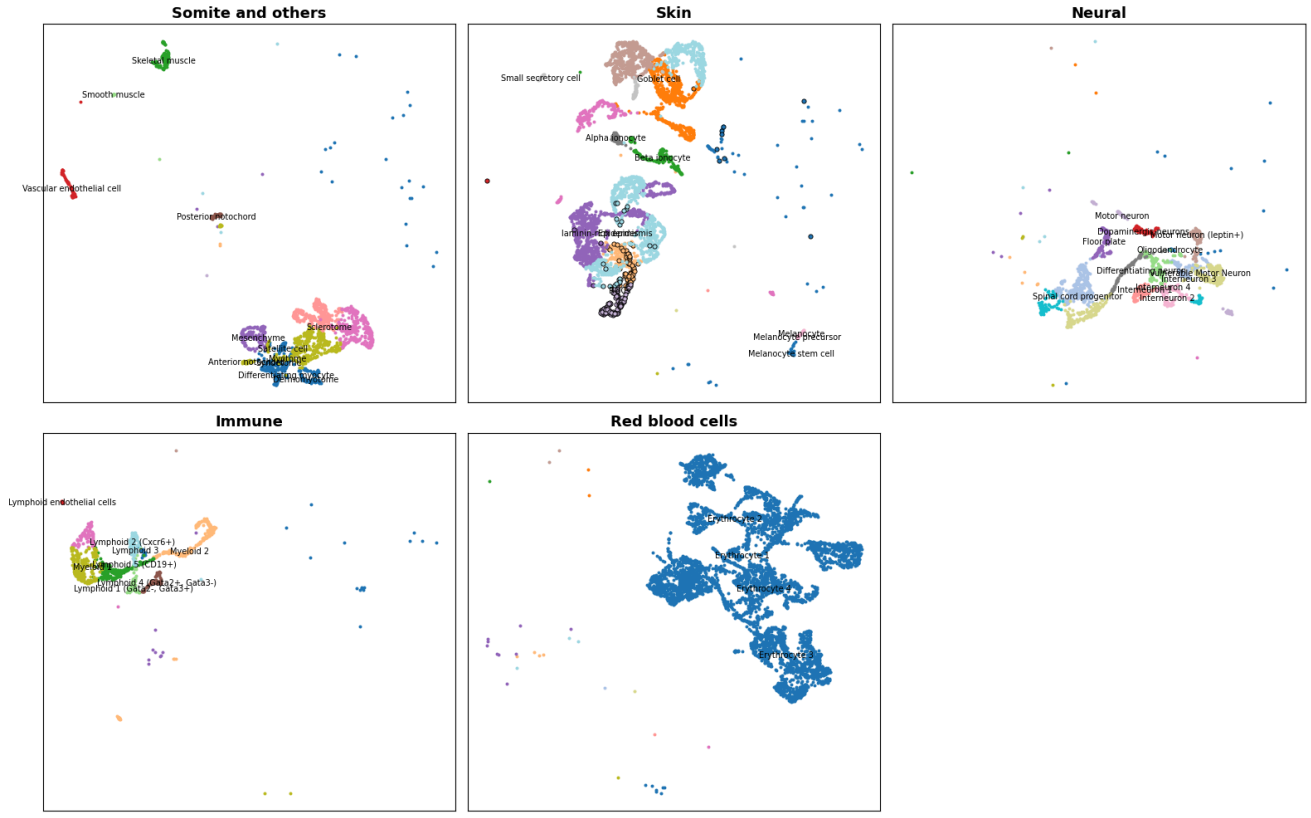



```
import numpy as np
import matplotlib.pyplot as plt
import scanpy as sc

# 1. Map clusters -> tissue categories on MAGIC

if "cluster" not in adata_magic.obs:
    raise ValueError("adata_magic.obs['cluster'] missing (needed for tissue categories).")

adata_magic.obs["category"] = (
    adata_magic.obs["cluster"]
    .map(cluster_to_category)
    .astype("category")
)

print("Category counts on MAGIC:")
print(adata_magic.obs["category"].value_counts())


# 2. Paper-style neighbors + UMAP on MAGIC
# (cosine, k=20, min_dist=0.5)

sc.pp.neighbors(
    adata_magic,
    n_neighbors=20,
    metric="cosine",
    method="umap",
    key_added="paper20"
)

sc.tl.umap(
    adata_magic,
```

```
    min_dist=0.5,
    neighbors_key="paper20"
)

X = adata_magic.obsm["X_umap"][:, 0]
Y = adata_magic.obsm["X_umap"][:, 1]

# 3. Colors for magic_kmeans clusters

if "magic_kmeans" not in adata_magic.obs:
    raise ValueError("adata_magic.obs['magic_kmeans'] missing (run MAGIC clustering first).")

km_labels = adata_magic.obs["magic_kmeans"].astype("category")
km_cats = km_labels.cat.categories

cmap = plt.get_cmap("tab20")
cluster_colors = {
    cl: cmap(i / max(1, len(km_cats) - 1))
    for i, cl in enumerate(km_cats)
}

# 4. Multi-panel figure by tissue category
panel_order = [
    ("Somites", "Somite and others"),
    ("Skin", "Skin"),
    ("Neural", "Neural"),
    ("Immune system", "Immune"),
    ("Red blood cells", "Red blood cells"),
]

fig, axes = plt.subplots(2, 3, figsize=(16, 10))
axes = axes.flatten()

# turn off any unused axes
for ax in axes[len(panel_order):]:
    ax.axis("off")

for (cat, title), ax in zip(panel_order, axes):
    mask_cat = (adata_magic.obs["category"] == cat).to_numpy()

    if not np.any(mask_cat):
        ax.axis("off")
        continue

    # clusters present in this tissue (based on original cluster labels)
    clusters_cat = (
        adata_magic.obs.loc[mask_cat, "cluster"]
        .astype("category")
        .cat.remove_unused_categories()
        .cat.categories
    )

    # plot each magic_kmeans cluster within this category
    for km_cl in km_cats:
        m = mask_cat & (km_labels == km_cl).to_numpy()
        if not np.any(m):
            continue

        # highlight ROC *cells* (based on original annotation) with black edge
        is_roc = (adata_magic.obs["cluster"] == "ROCs").to_numpy() & m

        # non-ROC cells of this kmeans cluster
        non_roc = m & ~is_roc

        if np.any(non_roc):
            ax.scatter(
                X[non_roc],
                Y[non_roc],
                s=8,
                c=[cluster_colors[km_cl]],
                linewidths=0,
            )

        if np.any(is_roc):
            ax.scatter(
                X[is_roc],
                Y[is_roc],
```

```
                    s=14,
                    c=[cluster_colors[km_cl]],
                    edgecolors="black",
                    linewidths=0.6,
                )

        # label original named clusters at their medians (only within this tissue)
        for cl in clusters_cat:
            m_lab = mask_cat & (adata_magic.obs["cluster"] == cl).to_numpy()
            if not np.any(m_lab):
                continue
            xm = np.median(X[m_lab])
            ym = np.median(Y[m_lab])
            ax.text(
                xm,
                ym,
                cl,
                fontsize=7,
                ha="center",
                va="center",
            )

        ax.set_title(title, fontsize=13, fontweight="bold")
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_frame_on(True)

    plt.tight_layout()
    plt.show()
```

```
Category counts on MAGIC:
category
Red blood cells    4737
Skin               4212
Somites            1716
Neural             1539
Immune system       995
Name: count, dtype: int64
```

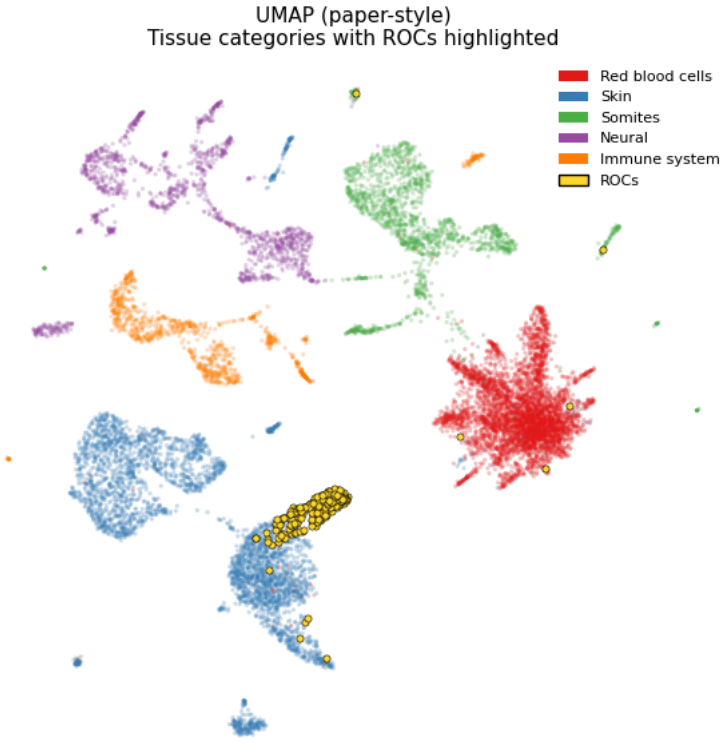



```
import igraph as ig
import matplotlib.patches as mpatches

# ========= 1. Start from raw =========
adata = adata_raw.copy()
adata.layers["counts"] = adata.X.copy()  # keep counts for other analyses

# ========= 2. Paper-style normalization + log2 =========
sc.pp.normalize_total(adata, target_sum=1e4)

if sp.issparse(adata.X):
    adata.X = adata.X.toarray()
adata.X = np.log2(adata.X + 1)

# ========= 3. HVG selection (Fano > 65th pct; mean 5–80th pct within HVGs) =========
Xlog = adata.X
means = Xlog.mean(axis=0)
vars_ = Xlog.var(axis=0, ddof=1)

fano = np.divide(vars_, means, out=np.zeros_like(vars_), where=means > 0)

thr = np.percentile(fano, 65)
hvg_mask = fano > thr
```

```
low, high = np.percentile(means[hvg_mask], [5, 80])
keep = hvg_mask & (means >= low) & (means <= high)

adata = adata[:, keep].copy()

# drop cells with all-zero across selected HVGs
cell_nonzero = (adata.X > 0).sum(axis=1) > 0
adata = adata[cell_nonzero, :].copy()

print("After paper-style HVG & filters:", adata)

#4. Clustering graph: fuzzy_simplicial_set (k=10) + Walktrap
sc.pp.neighbors(
    adata,
    n_neighbors=10,
    metric="cosine",
    method="umap",
    key_added="neighbors10",
)

C = adata.obsp["neighbors10_connectivities"]
if not sp.isspmatrix_csr(C):
    C = C.tocsr()

rows, cols = C.nonzero()
weights = np.asarray(C[rows, cols]).ravel()

g = ig.Graph(
    n=C.shape[0],
    edges=list(zip(rows, cols)),
    directed=False,
)
g.es["weight"] = weights

wt = g.community_walktrap(steps=10, weights="weight").as_clustering()
adata.obs["walktrap"] = pd.Categorical(wt.membership)

print("Walktrap clustering done. #clusters:", adata.obs["walktrap"].nunique())

#  5. UMAP for visualization (k=20, cosine, min_dist=0.5)
sc.pp.neighbors(
    adata,
    n_neighbors=20,
    metric="cosine",
    method="umap",
    key_added="neighbors20",
)

sc.tl.umap(
    adata,
    min_dist=0.5,
    neighbors_key="neighbors20",
)

#  6. Categories + ROC mask
if "cluster" not in adata.obs:
    raise ValueError("Expected 'cluster' in adata.obs to identify ROCs.")

adata.obs["category"] = adata.obs["cluster"].map(cluster_to_category)
# leave as object, NOT categorical, to avoid fillna issues
adata.obs["category"] = adata.obs["category"].astype(object)

roc_mask = (adata.obs["cluster"] == "ROCs").values
coords = adata.obsm["X_umap"]
categories = adata.obs["category"]

#  7. Single figure: tissue categories + ROC highlight

cat_colors = {
    "Red blood cells":  "#E41A1C",
    "Skin":             "#377EB8",
    "Somites":          "#4DAF4A",
    "Neural":           "#984EA3",
    "Immune system":    "#FF7F00",
}
default_color = "#BFBFBF"
```

```
bg_colors = categories.map(cat_colors).fillna(default_color).values

plt.figure(figsize=(6, 6))

plt.scatter(
    coords[:, 0],
    coords[:, 1],
    s=5,
    c=bg_colors,
    linewidths=0,
    alpha=0.25,
)

# overlay: ROC cells
plt.scatter(
    coords[roc_mask, 0],
    coords[roc_mask, 1],
    s=14,
    c="#FFD92F",
    edgecolors="#000000",
    linewidths=0.4,
    alpha=0.98,
)

plt.axis("off")
plt.title("UMAP (paper-style)\nTissue categories with ROCs highlighted", fontsize=11)

# legend
handles = [
    mpatches.Patch(facecolor=color, edgecolor="none", label=cat)
    for cat, color in cat_colors.items()
]
handles.append(mpatches.Patch(facecolor="#FFD92F", edgecolor="#000000", label="ROCs"))

plt.legend(handles=handles, frameon=False, loc="upper right", fontsize=8)

plt.tight_layout()
plt.show()
```

```
After paper-style HVG & filters: AnnData object with n_obs × n_vars = 13199 × 8277
    obs: 'barcode_cells', 'cell', 'sample', 'DevelopmentalStage', 'DaysPostAmputation', 'cluster', 'X', 'Y', 'CellCyc
    layers: 'counts'
Walktrap clustering done. #clusters: 92
```

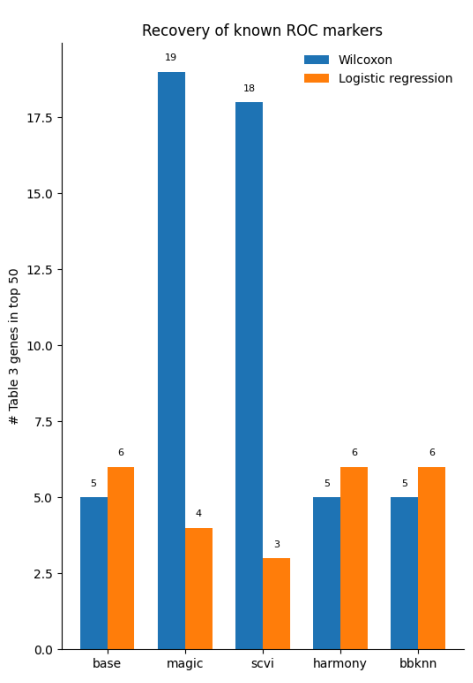


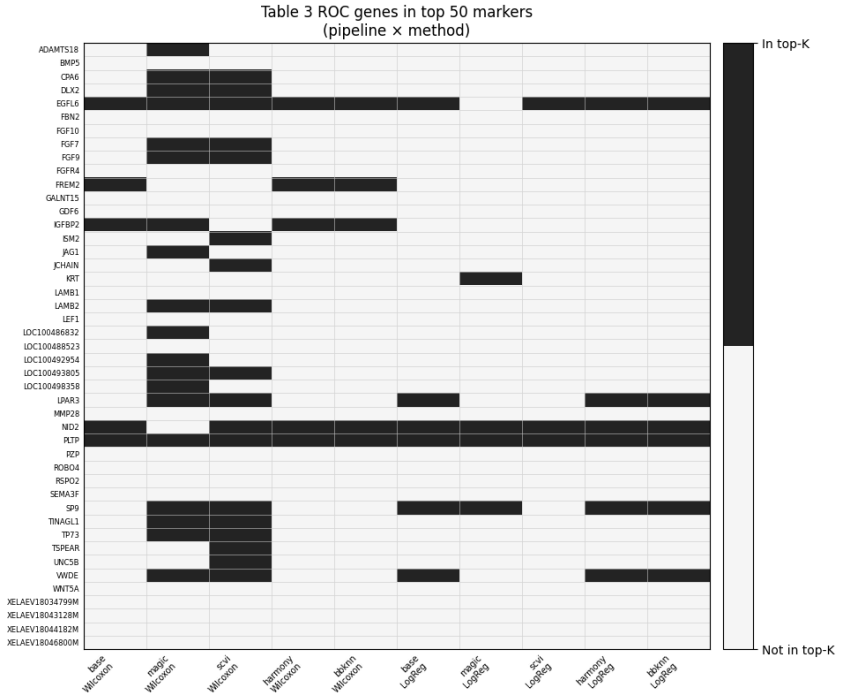
```
pipelines = ["base", "magic", "scvi", "harmony", "bbknn"]
pipeline_markers = {
    "base":    (base_roc_w, base_roc_l),
    "magic":   (magic_roc_w, magic_roc_l),
    "scvi":    (scvi_roc_w, scvi_roc_l),
    "harmony": (harmony_roc_w, harmony_roc_l),
    "bbknn":   (bbknn_roc_w, bbknn_roc_l),
}
```

Final Clustering Results

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# ===== config =====
K = 50  # top-K ROC markers to check
pipelines = ["base", "magic", "scvi", "harmony", "bbknn"]

# Assumes you have:
# pipeline_markers = {
#   "base":    (base_roc_w,    base_roc_l),
#   "magic":   (magic_roc_w,   magic_roc_l),
#   "scvi":    (scvi_roc_w,    scvi_roc_l),
#   "harmony": (harmony_roc_w, harmony_roc_l),
#   "bbknn":   (bbknn_roc_w,   bbknn_roc_l),
# }

def topK_table3_hits(roc_df, K=50):
    sub = roc_df.head(K).copy()
    sub["gene_c"] = sub["gene"].map(canon)
    genes = {g for g in sub["gene_c"] if g in TABLE3_CANON}
    return len(genes), genes

# ===== collect stats =====
records = []
heatmap_members = {}  # (pipeline, method) -> set of Table3 genes (canonical) in top-K

for p in pipelines:
    roc_w, roc_l = pipeline_markers[p]

    # Wilcoxon
    w_count, w_genes = topK_table3_hits(roc_w, K)
    records.append({"pipeline": p, "method": "Wilcoxon", "hits": w_count})
    heatmap_members[(p, "Wilcoxon")] = w_genes

    # LogReg
    l_count, l_genes = topK_table3_hits(roc_l, K)
    records.append({"pipeline": p, "method": "LogReg", "hits": l_count})
    heatmap_members[(p, "LogReg")] = l_genes

bar_df = pd.DataFrame(records)

# Prepare values in consistent order
x = np.arange(len(pipelines))
width = 0.35

wilcoxon_vals = [
    bar_df[(bar_df.pipeline == p) & (bar_df.method == "Wilcoxon")]["hits"].values[0]
    for p in pipelines
]
logreg_vals = [
    bar_df[(bar_df.pipeline == p) & (bar_df.method == "LogReg")]["hits"].values[0]
    for p in pipelines
]

# ===== build binary heatmap matrix =====
t3_genes_sorted = sorted(TABLE3_CANON)
cols = [(p, "Wilcoxon") for p in pipelines] + [(p, "LogReg") for p in pipelines]

heat = np.zeros((len(t3_genes_sorted), len(cols)), dtype=int)
for j, (p, m) in enumerate(cols):
    genes_here = heatmap_members[(p, m)]
    for i, g in enumerate(t3_genes_sorted):
```

```
            if g in genes_here:
                heat[i, j] = 1

    # ===== plotting: side-by-side =====
    fig, (ax_bar, ax_heat) = plt.subplots(
        1, 2, figsize=(15, max(4, len(t3_genes_sorted) * 0.18)),
        gridspec_kw={"width_ratios": [1.2, 2.0]}
    )

    # --- Left: grouped bar chart ---
    ax_bar.bar(x - width/2, wilcoxon_vals, width, label="Wilcoxon")
    ax_bar.bar(x + width/2, logreg_vals, width, label="Logistic regression")

    ax_bar.set_xticks(x)
    ax_bar.set_xticklabels(pipelines, rotation=0)
    ax_bar.set_ylabel(f"# Table 3 genes in top {K}")
    ax_bar.set_title("Recovery of known ROC markers")
    ax_bar.legend(frameon=False)

    # Add value labels for clarity
    for xi, v in zip(x - width/2, wilcoxon_vals):
        ax_bar.text(xi, v + 0.3, str(v), ha="center", va="bottom", fontsize=8)
    for xi, v in zip(x + width/2, logreg_vals):
        ax_bar.text(xi, v + 0.3, str(v), ha="center", va="bottom", fontsize=8)

    ax_bar.spines["top"].set_visible(False)
    ax_bar.spines["right"].set_visible(False)

    # --- Right: binary heatmap ---
    # Discrete 0/1 colormap: light for 0, dark for 1
    cmap = ListedColormap(["#f5f5f5", "#252525"])

    im = ax_heat.imshow(heat, aspect="auto", interpolation="nearest",
                        cmap=cmap, vmin=0, vmax=1)

    # Ticks & labels
    ax_heat.set_yticks(np.arange(len(t3_genes_sorted)))
    ax_heat.set_yticklabels(t3_genes_sorted, fontsize=6)
    ax_heat.set_xticks(np.arange(len(cols)))
    ax_heat.set_xticklabels(
        [f"{p}\n{m}" for (p, m) in cols],
        rotation=45,
        ha="right",
        fontsize=7
    )

    ax_heat.set_title(f"Table 3 ROC genes in top {K} markers\n(pipeline × method)")

    # Add grid lines to make it clearly gridded
    ax_heat.set_xticks(np.arange(-0.5, len(cols), 1), minor=True)
    ax_heat.set_yticks(np.arange(-0.5, len(t3_genes_sorted), 1), minor=True)
    ax_heat.grid(which="minor", color="#d0d0d0", linestyle="-", linewidth=0.4)
    ax_heat.tick_params(which="both", length=0)

    # Colorbar with binary label
    cbar = fig.colorbar(im, ax=ax_heat, fraction=0.046, pad=0.02)
    cbar.set_ticks([0, 1])
    cbar.set_ticklabels(["Not in top-K", "In top-K"])

    plt.tight_layout()
    plt.show()
```





For marker selection Wilcoxon + Logistic Regression

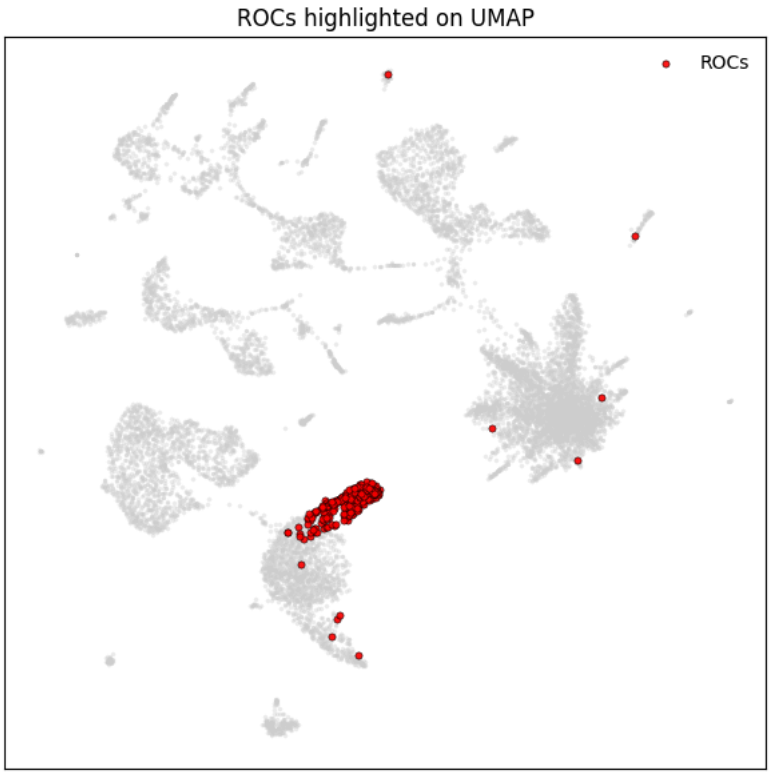
```
print(adata.obs.columns)

print("\ncluster value_counts():")
print(adata.obs["cluster"].value_counts())
```

```
Index(['barcode_cells', 'cell', 'sample', 'DevelopmentalStage',
       'DaysPostAmputation', 'cluster', 'X', 'Y', 'CellCyclePhase', 'Sample',
       'Lane', 'Condition', 'batch', 'walktrap', 'category'],
      dtype='object')

cluster value_counts():
cluster
Erythrocyte 1                2761
Epidermis                    1800
Goblet cell                  1754
Erythrocyte 3                 982
Erythrocyte 4                 862
Sclerotome                    752
Spinal cord progenitor        505
Myeloid 1                     475
Myotome                       303
Vulnerable Motor Neuron       284
Myeloid 2                     265
ROCs                          254
Mesenchyme                    215
Beta ionocyte                 207
Differentiating neuron        158
Erythrocyte 2                 132
Interneuron 1                 128
Skeletal muscle               117
Floor plate                   109
Dermomyotome                   84
Dopaminergic neurons           80
Lymphoid 4 (Gata2+, Gata3-)    77
Motor neuron                   76
Vascular endothelial cell      75
Interneuron 2                  73
Motor neuron (leptin+)         71
```

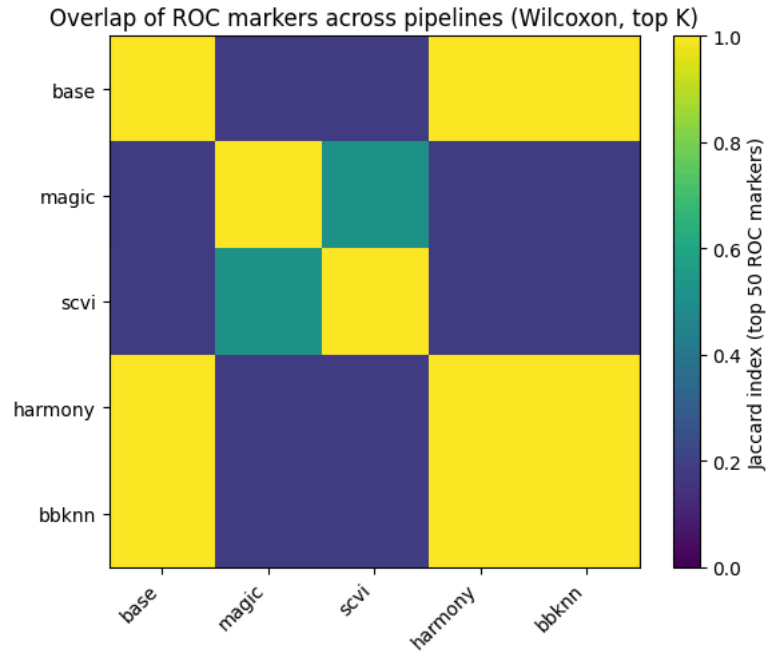
```
Lymphoid 2 (Cxcr6+)              65
Lymphoid 1 (Gata2−, Gata3+)      56
Alpha ionocyte                   56
Posterior notochord              54
Satellite cell                   44
Small secretory cell             37
Melanocyte stem cell             37
Anterior notochord               34
Lymphoid 3                       31
Melanocyte                       25
Interneuron 3                    24
Interneuron 4                    23
Melanocyte precursor             21
laminin-rich epidermis           21
Syndetome                        15
Lymphoid 5 (CD19+)               14
Lymphoid endothelial cells       12
Differentiating myocyte          12
Smooth muscle                    11
Oligodendrocyte                   8
Name: count, dtype: int64
```

```
import matplotlib.pyplot as plt

# Boolean mask for ROC cells
roc_mask = adata.obs["cluster"] == "ROCs"

print("ROC cells:", roc_mask.sum())
print("Total cells:", adata.n_obs)

# Needs adata.obsm["X_umap"] precomputed
coords = adata.obsm["X_umap"]

plt.figure(figsize=(6, 6))

# all cells (grey)
plt.scatter(
    coords[:, 0],
    coords[:, 1],
    s=5,
    c="#d0d0d0",
    alpha=0.4,
    linewidths=0,
)

# ROCs (red w/ outline)
plt.scatter(
    coords[roc_mask, 0],
    coords[roc_mask, 1],
    s=14,
    c="red",
    edgecolors="black",
    linewidths=0.4,
    alpha=0.9,
    label="ROCs",
)

plt.title("ROCs highlighted on UMAP")
plt.xticks([]); plt.yticks([])
plt.legend(frameon=False)
plt.tight_layout()
plt.show()
```

```
ROC cells: 254
Total cells: 13199
```




```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

K = 50  # top-K markers to compare



def topK_set(df, K):
    return set(df["gene"].head(K))

# Build Jaccard similarity matrix
jac = np.zeros((len(pipelines), len(pipelines)))

for i, p1 in enumerate(pipelines):
    s1 = topK_set(pipeline_markers[p1], K)
    for j, p2 in enumerate(pipelines):
        s2 = topK_set(pipeline_markers[p2], K)
        inter = len(s1 & s2)
        union = len(s1 | s2) if len(s1 | s2) > 0 else 1
        jac[i, j] = inter / union

plt.figure(figsize=(6, 5))
im = plt.imshow(jac, vmin=0, vmax=1, interpolation="nearest")
plt.colorbar(im, label=f"Jaccard index (top {K} ROC markers)")
plt.xticks(range(len(pipelines)), pipelines, rotation=45, ha="right")
plt.yticks(range(len(pipelines)), pipelines)
plt.title("Overlap of ROC markers across pipelines (Wilcoxon, top K)")
plt.tight_layout()
plt.show()
```



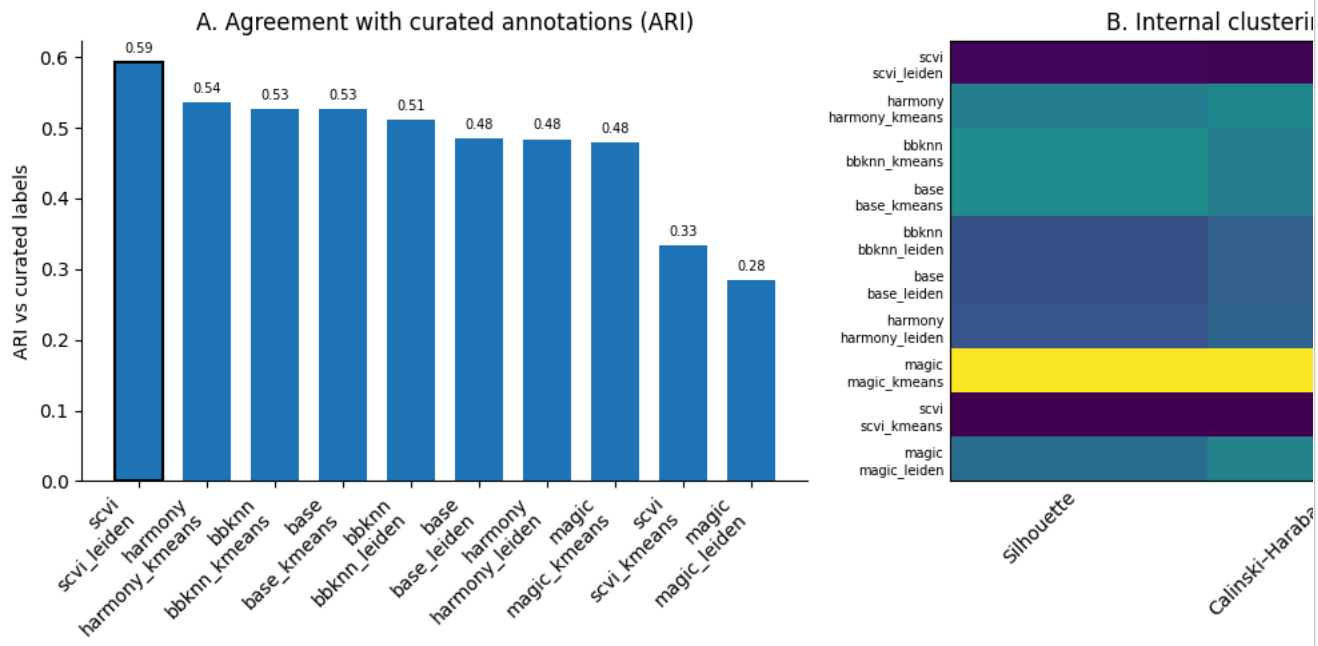

```
K = 50

# collect top-K sets per pipeline
top_sets = {}
for p in pipelines:
    roc_w, roc_l = pipeline_markers[p]
    # Combine genes from both Wilcoxon and LogReg top-K results
    combined_genes = set(roc_w["gene"].head(K)) | set(roc_l["gene"].head(K))
    top_sets[p] = {canon(g) for g in combined_genes}

all_genes = sorted(set().union(*top_sets.values()))

rows = []
for g in all_genes:
    present_in = [p for p in pipelines if g in top_sets[p]]
    rows.append({
        "gene_canon": g,
        "n_pipelines": len(present_in),
        "pipelines": ",".join(present_in),
        "in_Table3": g in TABLE3_CANON,
    })

consensus_df = pd.DataFrame(rows)

# sort: most robust first, Table3 hits at top
consensus_df = consensus_df.sort_values(
    by=["n_pipelines", "in_Table3"],
    ascending=[False, False]
).reset_index(drop=True)

# show top 30 for the report; keep full df for analysis
consensus_df.head(50)
```

Next steps: Generate code with consensus_df New interactive sheet

	gene_canon	n_pipelines	pipelines	in_Table3
0	EGFL6	5	base,magic,scvi,harmony,bbknn	True
1	LPAR3	5	base,magic,scvi,harmony,bbknn	True
2	NID2	5	base,magic,scvi,harmony,bbknn	True

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df = metrics_all.copy()

# Build a compact label per configuration
df["label"] = df["pipeline"].astype(str) + "\n" + df["method"].astype(str)

# Sort by ARI for cleaner visualization
df = df.sort_values("ARI_vs_ref", ascending=False).reset_index(drop=True)

labels = df["label"].values
ari = df["ARI_vs_ref"].values

# Internal metrics
sil = df["Silhouette"].values
ch = df["CalinskiHarabasz"].values
db = df["DaviesBouldin"].values

def norm(v):
    vmin, vmax = np.min(v), np.max(v)
    if vmax == vmin:
        return np.ones_like(v) * 0.5
    return (v - vmin) / (vmax - vmin)

sil_n = norm(sil)
ch_n = norm(ch)
db_inv_n = norm(db.max() - db)  # invert DB so higher = better

metric_names = ["Silhouette", "Calinski–Harabasz", "1 / Davies–Bouldin"]
metric_matrix = np.vstack([sil_n, ch_n, db_inv_n]).T

# plotting
fig, (ax_bar, ax_heat) = plt.subplots(
    1, 2,
    figsize=(14, 5),
    gridspec_kw={"width_ratios": [1.4, 1.6]}
)

# Panel A: ARI vs  labels
x = np.arange(len(labels))
bars = ax_bar.bar(x, ari, width=0.7)

# highlight best ARI
best_idx = np.argmax(ari)
bars[best_idx].set_edgecolor("black")
bars[best_idx].set_linewidth(1.5)

ax_bar.set_xticks(x)
ax_bar.set_xticklabels(labels, rotation=45, ha="right")
ax_bar.set_ylabel("ARI vs curated labels")
ax_bar.set_title("A. Agreement with curated annotations (ARI)")

ax_bar.spines["top"].set_visible(False)
ax_bar.spines["right"].set_visible(False)

# add ARI value labels
for i, v in enumerate(ari):
    ax_bar.text(i, v + 0.01, f"{v:.2f}", ha="center", va="bottom", fontsize=7)

# Panel B: Normalized internal metrics heatmap
im = ax_heat.imshow(
    metric_matrix,
    aspect="auto",
    interpolation="nearest",
    cmap="viridis",
    vmin=0,
    vmax=1,
```

```
)

ax_heat.set_yticks(np.arange(len(labels)))
ax_heat.set_yticklabels(labels, fontsize=7)
ax_heat.set_xticks(np.arange(len(metric_names)))
ax_heat.set_xticklabels(metric_names, rotation=45, ha="right")

ax_heat.set_title("B. Internal clustering quality (normalized)")
ax_heat.tick_params(axis="x", length=0)
ax_heat.tick_params(axis="y", length=0)

cbar = fig.colorbar(im, ax=ax_heat, fraction=0.046, pad=0.02)
cbar.set_label("Normalized score (higher = better)")

plt.tight_layout()
plt.show()
```




```
print(pipeline_markers)
```

```
196         aplp2.S  13.005704  1.882178   1.775344e-36
197          ier2.L  13.005588  1.732424   1.775344e-36
198        rab11a.S  13.002254  1.605552   1.845339e-36
199         lamb2.L  13.001680  5.924829   1.850121e-36

[200 rows x 4 columns], 'magic':                  gene      score     logFC       pval_adj
0              lpar3.L  26.772797  5.801023  1.753888e-153
1    Xetrov90029035m.L  26.711340  7.738851  4.546935e-153
2              lamb2.L  26.686167  5.630384  5.942070e-153
3       loc100493805.L  26.599154  6.087032  4.541773e-152
4    Xelaev18043128m.g  26.586615  7.036039  5.073594e-152
..                 ...        ...       ...           ...
195          lmx1b.1.L  25.028521  3.051035  3.994275e-136
196            mus81.L  25.027855  2.679368  4.040814e-136
197           kif26a.L  25.026192  3.397247  4.191521e-136
198           rassf3.S  25.025743  2.953377  4.217655e-136
199           atp8b1.L  25.024363  2.792796  4.344275e-136

[200 rows x 4 columns], 'scvi':                  gene      score     logFC       pval_adj
0    Xetrov90029035m.L  26.877703  6.560441  1.047511e-154
1              lamb2.L  26.801929  4.723192  3.796655e-154
2              egfl6.L  26.779732  4.917590  3.796655e-154
3               mmp3.L  26.778168  5.712176  3.796655e-154
```

```
[200 rows x 4 columns], 'harmony':                gene      score     logFC       pval_adj
0         col14a1.S  25.357527  5.884367  2.340832e-137
1      apoc1.like.L  25.222475  8.242413  3.580654e-136
2    loc100486548.L  25.060778  5.087452  1.400259e-134
3             mdk.L  24.477623  4.950447  2.016198e-128
4             id3.S  23.945599  4.214293  6.476695e-123
..              ...        ...       ...            ...
195          dlx5.L  13.019322  3.325606   1.498074e-36
196         aplp2.S  13.005704  1.882178   1.775344e-36
197          ier2.L  13.005588  1.732424   1.775344e-36
198        rab11a.S  13.002254  1.605552   1.845339e-36
199         lamb2.L  13.001680  5.924829   1.850121e-36

[200 rows x 4 columns], 'bbknn':                gene      score     logFC       pval_adj
0         col14a1.S  25.357527  5.884367  2.340832e-137
1      apoc1.like.L  25.222475  8.242413  3.580654e-136
2    loc100486548.L  25.060778  5.087452  1.400259e-134
3             mdk.L  24.477623  4.950447  2.016198e-128
4             id3.S  23.945599  4.214293  6.476695e-123
..              ...        ...       ...            ...
195          dlx5.L  13.019322  3.325606   1.498074e-36
196         aplp2.S  13.005704  1.882178   1.775344e-36
197          ier2.L  13.005588  1.732424   1.775344e-36
198        rab11a.S  13.002254  1.605552   1.845339e-36
199         lamb2.L  13.001680  5.924829   1.850121e-36

[200 rows x 4 columns]}
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

pipelines = ["base", "magic", "scvi", "harmony", "bbknn"]
# pipeline_markers = { "base": (base_roc_w, base_roc_l), ... }  # as before

def topK_table3_hits(roc_df, K):
    sub = roc_df.head(K).copy()
    sub["gene_c"] = sub["gene"].map(canon)
    genes = {g for g in sub["gene_c"] if g in TABLE3_CANON}
    return len(genes), genes

def plot_table3_overlap(K):
    # ---- collect stats ----
    records = []
    heatmap_members = {}

    for p in pipelines:
        roc_w, roc_l = pipeline_markers[p]

        # Wilcoxon
        w_count, w_genes = topK_table3_hits(roc_w, K)
        records.append({"pipeline": p, "method": "Wilcoxon", "hits": w_count})
        heatmap_members[(p, "Wilcoxon")] = w_genes

        # LogReg
        l_count, l_genes = topK_table3_hits(roc_l, K)
        records.append({"pipeline": p, "method": "LogReg", "hits": l_count})
        heatmap_members[(p, "LogReg")] = l_genes

    bar_df = pd.DataFrame(records)

    # bar values
    x = np.arange(len(pipelines))
    width = 0.35
    wilcoxon_vals = [
        bar_df[(bar_df.pipeline == p) & (bar_df.method == "Wilcoxon")]["hits"].iloc[0]
        for p in pipelines
    ]
    logreg_vals = [
        bar_df[(bar_df.pipeline == p) & (bar_df.method == "LogReg")]["hits"].iloc[0]
        for p in pipelines
    ]

    #binary heatmap matrix
    t3_genes_sorted = sorted(TABLE3_CANON)
    cols = [(p, "Wilcoxon") for p in pipelines] + [(p, "LogReg") for p in pipelines]

    heat = np.zeros((len(t3_genes_sorted), len(cols)), dtype=int)
    for j, (p, m) in enumerate(cols):
```