```
# This Python 3 environment comes with many helpful analytics
libraries installed
# It is defined by the kaggle/python Docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/"
directory
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/)
that gets preserved as output when you create a version using "Save &
Run All"
# You can also write temporary files to /kaggle/temp/, but they won't
be saved outside of the current session
```

```
/kaggle/input/breast-cancer-dataset10x-genomics/brca-R.npy
/kaggle/input/breast-cancer-dataset10x-genomics/brca-vi.h5ad
/kaggle/input/breast-cancer-dataset10x-genomics/brca-xe.h5ad
/kaggle/input/xen-imputed/best_imputed_xenium.npy
```

```
!pip install scanpy anndata
```

```
Collecting scanpy
  Downloading scanpy-1.11.5-py3-none-any.whl.metadata (9.3 kB)
Collecting anndata
  Downloading anndata-0.12.6-py3-none-any.whl.metadata (10.0 kB)
Requirement already satisfied: h5py>=3.7.0 in
/usr/local/lib/python3.11/dist-packages (from scanpy) (3.14.0)
Requirement already satisfied: joblib in
/usr/local/lib/python3.11/dist-packages (from scanpy) (1.5.2)
Collecting legacy-api-wrap>=1.4.1 (from scanpy)
  Downloading legacy_api_wrap-1.5-py3-none-any.whl.metadata (2.2 kB)
Collecting matplotlib>=3.7.5 (from scanpy)
  Downloading matplotlib-3.10.7-cp311-cp311-
manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (11 kB)
Requirement already satisfied: natsort in
/usr/local/lib/python3.11/dist-packages (from scanpy) (8.4.0)
Requirement already satisfied: networkx>=2.7.1 in
/usr/local/lib/python3.11/dist-packages (from scanpy) (3.5)
Requirement already satisfied: numba!=0.62.0rc1,>=0.57.1 in
```

```
/usr/local/lib/python3.11/dist-packages (from scanpy) (0.60.0)
Requirement already satisfied: numpy>=1.24.1 in
/usr/local/lib/python3.11/dist-packages (from scanpy) (1.26.4)
Requirement already satisfied: packaging>=21.3 in
/usr/local/lib/python3.11/dist-packages (from scanpy) (25.0)
Requirement already satisfied: pandas>=1.5.3 in
/usr/local/lib/python3.11/dist-packages (from scanpy) (2.2.3)
Requirement already satisfied: patsy!=1.0.0 in
/usr/local/lib/python3.11/dist-packages (from scanpy) (1.0.1)
Requirement already satisfied: pynndescent>=0.5.13 in
/usr/local/lib/python3.11/dist-packages (from scanpy) (0.5.13)
Requirement already satisfied: scikit-learn>=1.1.3 in
/usr/local/lib/python3.11/dist-packages (from scanpy) (1.2.2)
Requirement already satisfied: scipy>=1.8.1 in
/usr/local/lib/python3.11/dist-packages (from scanpy) (1.15.3)
Collecting seaborn>=0.13.2 (from scanpy)
  Downloading seaborn-0.13.2-py3-none-any.whl.metadata (5.4 kB)
Collecting session-info2 (from scanpy)
  Downloading session_info2-0.2.3-py3-none-any.whl.metadata (3.4 kB)
Requirement already satisfied: statsmodels>=0.14.5 in
/usr/local/lib/python3.11/dist-packages (from scanpy) (0.14.5)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-
packages (from scanpy) (4.67.1)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.11/dist-packages (from scanpy) (4.15.0)
Requirement already satisfied: umap-learn>=0.5.6 in
/usr/local/lib/python3.11/dist-packages (from scanpy) (0.5.9.post2)
Collecting array-api-compat>=1.7.1 (from anndata)
  Downloading array_api_compat-1.12.0-py3-none-any.whl.metadata (2.5
kB)
Collecting zarr!=3.0.*,>=2.18.7 (from anndata)
  Downloading zarr-3.1.3-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.7.5-
>scanpy) (1.3.2)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.7.5-
>scanpy) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.7.5-
>scanpy) (4.59.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.7.5-
>scanpy) (1.4.8)
Requirement already satisfied: pillow>=8 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.7.5-
>scanpy) (11.3.0)
Requirement already satisfied: pyparsing>=3 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.7.5-
```

```
>scanpy) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.7.5-
>scanpy) (2.9.0.post0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in
/usr/local/lib/python3.11/dist-packages (from numba!
=0.62.0rc1,>=0.57.1->scanpy) (0.43.0)
Requirement already satisfied: mkl_fft in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.24.1->scanpy)
(1.3.8)
Requirement already satisfied: mkl_random in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.24.1->scanpy)
(1.2.4)
Requirement already satisfied: mkl_umath in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.24.1->scanpy)
(0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.11/dist-
packages (from numpy>=1.24.1->scanpy) (2025.3.0)
Requirement already satisfied: tbb4py in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.24.1->scanpy)
(2022.3.0)
Requirement already satisfied: mkl-service in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.24.1->scanpy)
(2.4.1)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.11/dist-packages (from pandas>=1.5.3->scanpy)
(2025.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.11/dist-packages (from pandas>=1.5.3->scanpy)
(2025.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.1.3-
>scanpy) (3.6.0)
Collecting scikit-learn>=1.1.3 (from scanpy)
  Downloading scikit_learn-1.7.2-cp311-cp311-
manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (11 kB)
Collecting donfig>=0.8 (from zarr!=3.0.*,>=2.18.7->anndata)
  Downloading donfig-0.8.1.post1-py3-none-any.whl.metadata (5.0 kB)
Collecting numcodecs>=0.14 (from numcodecs[crc32c]>=0.14->zarr!
=3.0.*,>=2.18.7->anndata)
  Downloading numcodecs-0.16.3-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.3 kB)
Requirement already satisfied: pyyaml in
/usr/local/lib/python3.11/dist-packages (from donfig>=0.8->zarr!
=3.0.*,>=2.18.7->anndata) (6.0.3)
Collecting crc32c>=2.7 (from numcodecs[crc32c]>=0.14->zarr!
=3.0.*,>=2.18.7->anndata)
  Downloading crc32c-2.8-cp311-cp311-
manylinux1_x86_64.manylinux_2_28_x86_64.manylinux_2_5_x86_64.whl.metad
```

```
ata (7.8 kB)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7-
>matplotlib>=3.7.5->scanpy) (1.17.0)
Requirement already satisfied: onemkl-license==2025.3.0 in
/usr/local/lib/python3.11/dist-packages (from mkl->numpy>=1.24.1-
>scanpy) (2025.3.0)
Requirement already satisfied: intel-openmp<2026,>=2024 in
/usr/local/lib/python3.11/dist-packages (from mkl->numpy>=1.24.1-
>scanpy) (2024.2.0)
Requirement already satisfied: tbb==2022.* in
/usr/local/lib/python3.11/dist-packages (from mkl->numpy>=1.24.1-
>scanpy) (2022.3.0)
Requirement already satisfied: tcmlib==1.* in
/usr/local/lib/python3.11/dist-packages (from tbb==2022.*->mkl-
>numpy>=1.24.1->scanpy) (1.4.0)
Requirement already satisfied: intel-cmplr-lib-rt in
/usr/local/lib/python3.11/dist-packages (from mkl_umath-
>numpy>=1.24.1->scanpy) (2024.2.0)
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in
/usr/local/lib/python3.11/dist-packages (from intel-
openmp<2026,>=2024->mkl->numpy>=1.24.1->scanpy) (2024.2.0)
Downloading scanpy-1.11.5-py3-none-any.whl (2.1 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 37.0 MB/s eta
0:00:00a 0:00:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 172.3/172.3 kB 12.7 MB/s eta
0:00:00
pat-1.12.0-py3-none-any.whl (58 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.2/58.2 kB 4.2 MB/s eta
0:00:00
atplotlib-3.10.7-cp311-cp311-
manylinux2014_x86_64.manylinux_2_17_x86_64.whl (8.7 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.7/8.7 MB 113.9 MB/s eta
0:00:0000:0100:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 294.9/294.9 kB 20.2 MB/s eta
0:00:00
anylinux2014_x86_64.manylinux_2_17_x86_64.whl (9.7 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.7/9.7 MB 98.3 MB/s eta
0:00:00:00:0100:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 276.4/276.4 kB 21.7 MB/s eta
0:00:00
codecs-0.16.3-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (8.9 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.9/8.9 MB 90.8 MB/s eta
0:00:00:00:0100:01
anylinux1_x86_64.manylinux_2_28_x86_64.manylinux_2_5_x86_64.whl (80
kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 80.2/80.2 kB 5.9 MB/s eta
0:00:00
```

```
pat, numcodecs, scikit-learn, zarr, matplotlib, seaborn, anndata,
scanpy
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.2.2
    Uninstalling scikit-learn-1.2.2:
      Successfully uninstalled scikit-learn-1.2.2
  Attempting uninstall: matplotlib
    Found existing installation: matplotlib 3.7.2
    Uninstalling matplotlib-3.7.2:
      Successfully uninstalled matplotlib-3.7.2
  Attempting uninstall: seaborn
    Found existing installation: seaborn 0.12.2
    Uninstalling seaborn-0.12.2:
      Successfully uninstalled seaborn-0.12.2
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
bigframes 2.12.0 requires google-cloud-bigquery-
storage<3.0.0,>=2.30.0, which is not installed.
ydata-profiling 4.17.0 requires matplotlib<=3.10,>=3.5, but you have
matplotlib 3.10.7 which is incompatible.
category-encoders 2.7.0 requires scikit-learn<1.6.0,>=1.0.0, but you
have scikit-learn 1.7.2 which is incompatible.
cesium 0.12.4 requires numpy<3.0,>=2.0, but you have numpy 1.26.4
which is incompatible.
sklearn-compat 0.1.3 requires scikit-learn<1.7,>=1.2, but you have
scikit-learn 1.7.2 which is incompatible.
bigframes 2.12.0 requires rich<14,>=12.4.4, but you have rich 14.2.0
which is incompatible.
Successfully installed anndata-0.12.6 array-api-compat-1.12.0 crc32c-
2.8 donfig-0.8.1.post1 legacy-api-wrap-1.5 matplotlib-3.10.7
numcodecs-0.16.3 scanpy-1.11.5 scikit-learn-1.7.2 seaborn-0.13.2
session-info2-0.2.3 zarr-3.1.3
```

```
# Load data from the separate files instead of BRCA.dmp
import numpy as np
import scanpy as sc

# Load the three data components from the 20250813 SIID Data Dump
folder
vi =
sc.read_h5ad("/kaggle/input/breast-cancer-dataset10x-genomics/brca-
vi.h5ad")
xe =
sc.read_h5ad("/kaggle/input/breast-cancer-dataset10x-genomics/brca-
xe.h5ad")
R = np.load("/kaggle/input/breast-cancer-dataset10x-genomics/brca-
R.npy")
```

```
# Apply the same preprocessing as the original code
vi.var_names_make_unique()
xe.var_names_make_unique()
if hasattr(vi.X, "toarray"): vi.X = vi.X.toarray()
if hasattr(xe.X, "toarray"): xe.X = xe.X.toarray()



# Verify the loaded data structure
print("Visium data (vi) shape:", vi.shape)
print("Xenium data (xe) shape:", xe.shape)
print("Transformation matrix (R) shape:", R.shape)
print("\nVisium genes (first 10):", list(vi.var_names[:10]))
print("Xenium genes (first 10):", list(xe.var_names[:10]))
print("Common genes:",
len(set(vi.var_names).intersection(set(xe.var_names))))


# VISIUM DATA VIEWER - Customizable dimensions
rows = 10  # Change this value to view different number of rows
cols = 10  # Change this value to view different number of columns

print("="*50)
print(f"VISIUM DATA ({rows}x{cols} values):")
print("="*50)
print(f"Total shape: {vi.shape}")
print(f"Row names (first {rows} observations):",
list(vi.obs_names[:rows]))
print(f"Column names (first {cols} genes):",
list(vi.var_names[:cols]))

# Set numpy print options to show full arrays without truncation
np.set_printoptions(threshold=np.inf, suppress=True)
print("Data values:")
print(vi.X[:rows, :cols])

# Display spatial coordinates if available
if 'spatial' in vi.obsm:
    print(f"\nVisium spatial coordinates (first {min(rows, 10)}):")
    print(vi.obsm['spatial'][:min(rows, 10)])

# Reset numpy print options to default
np.set_printoptions(threshold=1000, suppress=False)


# XENIUM DATA VIEWER - Customizable dimensions
rows = 10  # Change this value to view different number of rows
cols = 10  # Change this value to view different number of columns

print("="*50)
```

```
print(f"XENIUM DATA ({rows}x{cols} values):")
print("="*50)
print(f"Total shape: {xe.shape}")
print(f"Row names (first {rows} observations):",
list(xe.obs_names[:rows]))
print(f"Column names (first {cols} genes):",
list(xe.var_names[:cols]))

# Set numpy print options to show full arrays without truncation
np.set_printoptions(threshold=np.inf, suppress=True)
print("Data values:")
print(xe.X[:rows, :cols])

# Display spatial coordinates if available
if 'spatial' in xe.obsm:
    print(f"\nXenium spatial coordinates (first {min(rows, 10)}):")
    print(xe.obsm['spatial'][:min(rows, 10)])

# Reset numpy print options to default
np.set_printoptions(threshold=1000, suppress=False)

# TRANSFORMATION MATRIX R VIEWER - COMPLETE MATRIX
print("="*50)
print(f"COMPLETE TRANSFORMATION MATRIX R:")
print("="*50)
print(f"Total shape: {R.shape} (rows: Xenium cells, cols: Visium
spots)")

# Set numpy print options to show ENTIRE matrix without truncation
np.set_printoptions(threshold=np.inf, suppress=True, linewidth=200)

print("\nComplete R matrix data:")
print("Rows = Xenium cells, Columns = Visium spots")
print(R)

# Reset numpy print options to default
np.set_printoptions(threshold=1000, suppress=False, linewidth=75)

print(f"\nMatrix printed completely: {R.shape[0]} rows × {R.shape[1]}
columns")

# R MATRIX DIMENSIONS
print("="*50)
print("R MATRIX DIMENSIONS")
print("="*50)
print(f"R matrix shape: {R.shape}")
print(f"Number of rows (Xenium cells): {R.shape[0]}")
print(f"Number of columns (Visium spots): {R.shape[1]}")
print(f"Total elements: {R.size:,}")
print(f"Memory size: {R.nbytes / (1024**2):.2f} MB")
```

```
/usr/local/lib/python3.11/dist-packages/anndata/_core/anndata.py:1798:
UserWarning: Variable names are not unique. To make them unique, call
`.var_names_make_unique`.
  utils.warn_names_duplicates("var")

Visium data (vi) shape: (4992, 18085)
Xenium data (xe) shape: (167780, 313)
Transformation matrix (R) shape: (3, 3)

Visium genes (first 10): ['SAMD11', 'NOC2L', 'KLHL17', 'PLEKHN1',
'PERM1', 'HES4', 'ISG15', 'AGRN', 'RNF223', 'C1orf159']
Xenium genes (first 10): ['ABCC11', 'ACTA2', 'ACTG2', 'ADAM9',
'ADGRE5', 'ADH1B', 'ADIPOQ', 'AGR3', 'AHSP', 'AIF1']
Common genes: 307
==================================================
VISIUM DATA (10x10 values):
==================================================
Total shape: (4992, 18085)
Row names (first 10 observations): ['AACACCTACTATCGAA-1',
'AACACGTGCATCGCAC-1', 'AACACTTGGCAAGGAA-1', 'AACAGGAAGAGCATAG-1',
'AACAGGATTCATAGTT-1', 'AACAGGCCAACGATTA-1', 'AACAGGTTATTGCACC-1',
'AACAGGTTCACCGAAG-1', 'AACAGTCAGGCTCCGC-1', 'AACAGTCCACGCGGTG-1']
Column names (first 10 genes): ['SAMD11', 'NOC2L', 'KLHL17',
'PLEKHN1', 'PERM1', 'HES4', 'ISG15', 'AGRN', 'RNF223', 'C1orf159']
Data values:
[[ 0.  0.  0.  0.  0.  1.  0.  1.  0.  0.]
 [ 2.  0.  0.  0.  0.  0.  0.  4.  0.  0.]
 [ 1.  3.  0.  1.  0.  3. 21. 10.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  2.  2.  0.  0.]
 [ 0.  0.  0.  0.  0.  1.  1.  1.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  2.  1.  1.  0.  2.  9.  5.  0.  1.]
 [ 6.  7.  0.  2.  0.  3. 10. 13.  0.  1.]
 [ 1.  1.  0.  0.  0.  4.  5.  4.  1.  0.]]

Visium spatial coordinates (first 10):
[[2357.24249778 2100.4678081 ]
 [8460.85314131 6790.54649653]
 [6135.64392165 4497.37055498]
 [7890.35571195 7482.56699685]
 [6287.1664118  5798.5318043 ]
 [8094.75853424 1912.33585724]
 [4606.18200764 3789.07918324]
 [6448.34973186 5892.59777973]
 [4254.32441306 7500.8717272 ]
 [3286.71602797 7307.65512902]]
==================================================
XENIUM DATA (10x10 values):
==================================================
```

```
Total shape: (167780, 313)
Row names (first 10 observations): ['1', '2', '3', '4', '5', '6', '7',
'8', '9', '10']
Column names (first 10 genes): ['ABCC11', 'ACTA2', 'ACTG2', 'ADAM9',
'ADGRE5', 'ADH1B', 'ADIPOQ', 'AGR3', 'AHSP', 'AIF1']
Data values:
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 2. 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. 2.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 2. 1. 0. 0. 0. 0. 0.]]

Xenium spatial coordinates (first 10):
[[326.19136505 847.25991211]
 [328.03182983 826.34199524]
 [331.74318695 848.76691895]
 [334.25264282 824.22840881]
 [332.24250488 841.35753784]
 [336.5739563  848.02223511]
 [338.13569641 835.28458252]
 [341.71234741 828.72623901]
 [344.51147461 847.60309753]
 [345.30679321 838.76473694]]
==================================================
COMPLETE TRANSFORMATION MATRIX R:
==================================================
Total shape: (3, 3) (rows: Xenium cells, cols: Visium spots)

Complete R matrix data:
Rows = Xenium cells, Columns = Visium spots
[[    0.04898181    -1.08703629  8646.65212296]
 [    1.08703629     0.04898181 -2602.96034584]
 [    0.             0.             1.        ]]

Matrix printed completely: 3 rows × 3 columns
==================================================
R MATRIX DIMENSIONS
==================================================
R matrix shape: (3, 3)
Number of rows (Xenium cells): 3
Number of columns (Visium spots): 3
Total elements: 9
Memory size: 0.00 MB
```

```
# --- This is a new cell, run this before running the modified
training ---
# --- (This is the content of your old Cell 17) ---

import numpy as np
import pickle
import os.path

version = 2

score_metrics = ["r2", "pcc", "ssim", "rmse", "js", "nnz"]
score_metrics_vs_src = ["r2", "pcc", "ssim", "rmse", "js"]

def score_corr(x, y, metric = "r2"):
    # All-in-one scoring function. All of them are structured such
that larger = better correlation.
    # Flatten ndarray if needed
    if isinstance(x, np.ndarray):
        xf = x.flatten()
    else:
        xf = np.array(x)
    if isinstance(y, np.ndarray):
        yf = y.flatten()
    else:
        yf = np.array(y)
    assert len(xf) == len(yf)
    if (np.any(xf == 0)) or (np.any(yf == 0)):
        if metric == "js":
            xf = xf + 1e-10
            yf = yf + 1e-10
    if (np.std(xf) < 1e-5) or (np.std(yf) < 1e-5):
        if metric in ['r2', 'pcc', 'ssim']:
            return 0
        elif metric == 'rmse':
            return -1
    n = len(xf)
    # The following 5 metrics comes from the Nat Meth paper
    if metric == "r2":
        corr_matrix = np.corrcoef(xf, yf)
        return corr_matrix[0, 1]**2
    elif metric == "pcc":
        mx, my = np.mean(xf), np.mean(yf)
        a = np.mean((xf - mx) * (yf - my))
        b = np.std(xf) * np.std(yf)
        return a / b
    elif metric == "ssim":
        x_, y_ = xf / max(xf), yf / max(yf)
        mx, my = np.mean(x_), np.mean(y_)
        sx, sy = np.std(x_), np.std(y_)
        cov = np.cov(x_, y_)[0][1]
```

```
        c1 = 0.01
        c2 = 0.03
        return (2 * mx * my + c1 ** 2) * (2 * cov + c2 ** 2) / (mx **
2 + my ** 2 + c1 ** 2) / (sx ** 2 + sy ** 2 + c2 ** 2)
    elif metric == "rmse":
        mx, my = np.mean(xf), np.mean(yf)
        sx, sy = np.std(xf), np.std(yf)
        zx = (xf - mx) / sx
        zy = (yf - my) / sy
        avg = np.mean((zx - zy) ** 2)
        return - (avg ** 0.5)
    elif metric == "js":
        xf = xf / sum(xf)
        yf = yf / sum(yf)
        return - (np.sum(xf * np.log(xf / yf)) + np.sum(yf * np.log(yf
/ xf))) / 2
    elif metric == "nnz":
        # A custom metric. It works as follows: Identify non-zero
elements in x, and estimate the overlap between the indices
        # and the top-N indices in y (N = # nnz).
        # Different from everything else: requires x and y to be
ordered - x be the observation and y be the estimate.
        nnz = xf > 0.1
        count = sum(nnz)
        if count == 0:
            return 0  # x is all-zero, meaningless
        sorted_est = np.argsort(yf)
        overlap = 0
        for i in sorted_est[-count:]:
            overlap += nnz[i]
        return overlap / count

def avg_corr_by_row(gt, est, met = "r2", eps = 0):
    assert gt.shape == est.shape
    data = []
    for i in range(gt.shape[0]):
        data.append(score_corr(gt[i] + eps, est[i] + eps, metric =
met))
    return sum(data) / len(data)

def corr_by_col(gt, est, met = "r2", eps = 0):
    assert gt.shape == est.shape
    data = []
    for i in range(gt.shape[1]):
        data.append(score_corr(gt[:, i] + eps, est[:, i] + eps, metric
= met))
    return data

def avg_corr_by_col(gt, est, met = "r2", eps = 0):
    return np.mean(corr_by_col(gt, est, met, eps))
```

```
import numpy as np
from scipy.sparse import csr_matrix
from anndata import AnnData
from scipy.spatial import KDTree  # <-- THIS LINE IS THE FIX

def apply(proj, p):
    return proj.dot(np.concatenate((p, [1])))[:2]

def batch_apply(proj, ps):
    ret = np.zeros_like(ps)
    for i in range(ps.shape[0]):
        ret[i] = apply(proj, ps[i])
    return ret


def gen_gamma(vi_sp, xe_sp, max_radius = 100, exclusive = True):
    assert exclusive
    ret = np.zeros((xe_sp.shape[0], vi_sp.shape[0]))
    kdt = KDTree(vi_sp) # This will now work
    for i in range(xe_sp.shape[0]):
        d, vi_idx = kdt.query(xe_sp[i])
        if d <= max_radius:
            ret[i, vi_idx] = 1
    return ret

import torch
import torch.nn as nn
import torch.nn.functional as F
from anndata import AnnData
from scipy.sparse import csr_matrix, issparse
import numpy as np

# --- Helper functions (same as before) ---
def to_dense_mat(ad, eps):
    if issparse(ad.X):
        mat = ad.X.toarray()
    else:
        mat = ad.X
    return mat + eps

def convert_to_sparse_csr_tensor(array):
    if not issparse(array):
        array = csr_matrix(array)
    crow_indices = torch.tensor(array.indptr, dtype=torch.int64)
    col_indices = torch.tensor(array.indices, dtype=torch.int64)
    values = torch.tensor(array.data, dtype=torch.float32)
    sparse_csr_tensor = torch.sparse_csr_tensor(crow_indices,
col_indices, values, size=array.shape)
    return sparse_csr_tensor.to_sparse()
```

```python
# --- NEW HELPER FUNCTIONS TO FIX THE NameError ---
def has_nan(t):
    """Checks if a tensor contains any NaN values."""
    return torch.isnan(t).any()

def has_inf(t):
    """Checks if a tensor contains any Inf values."""
    return torch.isinf(t).any()

def dcn(t):
    """Detaches a tensor, moves it to CPU, and converts to NumPy."""
    return t.detach().cpu().numpy()
# ----------------------------------------


class LowDimWithScaling(torch.nn.Module):
    """
    This class finds out low-dimensional representation Given two
count matrices.
    The main workhorse behind the jointly optimizing membership
functions for the Xenium and visium
    datasets.
    This is adapted from Hongyu's code.
    Parameters
    ----------
    xe : AnnData
        The AnnData object for Xenium dataset
    vi : AnnData
        The AnnData object for Visium dataset
    gamma : float
        The mapping matrix that maps the Xenium cells to visium data,
this is obtained from spatial
        mapping.
    n_factors : int
        The number of cell-types
    device : str
        The device on which the model is run.
    eps : float
        The epsilon value for the model.
    """
    def __init__(
            self,
            xe: AnnData,
            vi: AnnData,
            gamma: np.ndarray,
            n_factors: int,
            lambda_x: float = 1.0,
            lambda_v: float = 1.0,
            lambda_r: float = 1.0,
            l2_w: float = 1e-5,
```

```
            entropy_w: float = 1.0,
            adaptive_entropy: bool = False,
            adaptive_entropy_denominator: int = 500,
            device: str = 'gpu0',
            eps: float = 1e-6,
            init_scale_xenium: bool = False,
            platform_scaling: bool = False,
            add_entropy: bool = False,
            poisson_sum: bool = False
        ):
        super().__init__()
        self.xe = xe
        self.vi = vi
        self.gamma = convert_to_sparse_csr_tensor(gamma).to(device)
        self.n_factors = n_factors
        self.lambda_x = lambda_x
        self.lambda_v = lambda_v
        self.lambda_r = lambda_r
        self.l2_w = l2_w
        self.entropy_w = entropy_w
        self.device = device
        self.num_xenium_cells = self.xe.shape[0]
        self.num_visium_cells = self.vi.shape[0]
        self.num_xenium_genes = self.xe.shape[1]
        self.num_genes = self.vi.shape[1]
        self.add_entropy = add_entropy
        self.adaptive_entropy = adaptive_entropy
        self.adaptive_entropy_denominator =
adaptive_entropy_denominator
        self.poisson_sum = poisson_sum

        # These are observed data
        self.X = torch.tensor(to_dense_mat(self.xe, 0),
dtype=torch.float32).to(device)
        self.V = torch.tensor(to_dense_mat(self.vi, 0),
dtype=torch.float32).to(device)

        # These are parameters to be learned
        self.Px = torch.nn.Parameter(
            torch.randn((self.num_xenium_cells, self.n_factors),
dtype=torch.float32).to(device) * 0.01
        )
        # This is not needed if you consider mapping matrix is
constant and
        # it's perfect
        # self.Pv = torch.nn.Parameter(
        #     torch.zeros((self.num_visium_cells, self.n_factors),
dtype=torch.float32).to(device))# what's the equivalent ?
        self.Q = torch.nn.Parameter(
```

```
            torch.randn((self.n_factors, self.num_genes),
dtype=torch.float32).to(device))
        # If we need scaling factor then
        # Each cell/spot has a separate scaling factor
        if init_scale_xenium:
            # adds eps to avoid log(0)
            self.scale_factor_xenium = torch.nn.Parameter(
                torch.tensor(np.log(self.xe.to_df().sum(1).values +
eps), dtype=torch.float33).view(-1,1).to(device))
        else:
            self.scale_factor_xenium = torch.nn.Parameter(
                torch.ones((self.num_xenium_cells, 1),
dtype=torch.float32).to(device))
        self.platform_scaling = platform_scaling
        self.platform_sf = torch.nn.Parameter(torch.randn((1,
self.num_xenium_genes), dtype = torch.float32).to(device))
        self.scale_factor_visium = torch.nn.Parameter(
            torch.ones((self.num_xenium_cells, 1),
dtype=torch.float32).to(device))

        # Initialize the learnable parameters Px, Pv, Q
        torch.nn.init.xavier_uniform_(self.Px)
        torch.nn.init.xavier_uniform_(self.Q)

    @property
    def Q_soft(self):
        return torch.nn.Softmax(dim=1)(self.Q)

    @property
    def Px_soft(self):
        return torch.nn.Softmax(dim=1)(self.Px)

    # Get the estimate of the Xenium counts
    def get_xenium_est(self, end_at=None):
        if end_at is None:
            end_at = self.num_genes

        #Q_T_soft = torch.nn.Softmax(dim=1)(self.Q_soft[:, :end_at])
        ret = torch.mm(self.Px_soft, self.Q_soft[:, :end_at])
        #ret = torch.mm(self.Px_soft, Q_T_soft)
        ret = torch.exp(self.scale_factor_xenium) * ret
        return ret

    # Get the estimate of the Visium counts
    def get_visium_est(self):
        scaled_Px = torch.exp(self.scale_factor_visium) * self.Px_soft
        combined_Px = torch.mm(self.gamma.T, scaled_Px)
        ret = torch.mm(combined_Px, self.Q_soft)
        if self.platform_scaling:
            ret[:, :self.xe.shape[1]] *= torch.exp(self.platform_sf)
```

```
        return ret

    def loss(self, verbose=False,loss_type='cosine',num_epoch=1):
        xe_est = self.get_xenium_est(end_at=self.num_xenium_genes)
        vi_est = self.get_visium_est()

        test_tensors = [xe_est, vi_est]
        for idx, t in enumerate(test_tensors):
            # --- THESE ARE THE FIXED LINES ---
            assert not has_nan(t), f"Training Failure: Tensor #{idx}
contains NaN"
            assert not has_inf(t), f"Training Failure: Tensor #{idx}
contains Inf"

        # Do cosine loss between X and xe_est
        if loss_type == 'cosine':
            x_loss = self.lambda_x * (2- \
                cosine_similarity(self.X, xe_est, dim=1).mean() -\
                cosine_similarity(self.X, xe_est, dim=0).mean()
            )
            v_loss = self.lambda_v * (2- \
                cosine_similarity(self.V, vi_est, dim=1).mean() -\
                cosine_similarity(self.V, vi_est, dim=0).mean()
            )
            expression_loss = (x_loss + v_loss)
        elif loss_type == 'kl':
            kl_loss = torch.nn.KLDivLoss(reduction='batchmean')
            xe_norm = F.log_softmax(xe_est, dim=1)
            target_x = F.softmax(self.X, dim=1)
            vi_norm = F.log_softmax(vi_est, dim=1)
            target_v = F.softmax(self.V, dim=1)
            test_tensors = [xe_norm, target_x, vi_norm, target_v]
            for idx, t in enumerate(test_tensors):
                # --- THESE ARE THE FIXED LINES ---
                assert not has_nan(t), f"Training Failure: Tensor
#{idx} contains NaN"
                assert not has_inf(t), f"Training Failure: Tensor
#{idx} contains Inf"
            x_loss = kl_loss(xe_norm, target_x)
            v_loss = kl_loss(vi_norm, target_v)
            expression_loss = x_loss + v_loss
        elif loss_type == 'fro':
            x_loss = self.lambda_x * torch.norm(self.X - xe_est, p =
'fro')**2
            v_loss = self.lambda_v * torch.norm(self.V - vi_est, p =
'fro')**2
            expression_loss = x_loss + v_loss
        elif loss_type == 'poisson':
            # no need to add eps here, poisson_nll_loss has intrinstic
```

```
eps term with log_input = False
            if self.poisson_sum:
                x_loss = self.lambda_x * F.poisson_nll_loss(xe_est,
self.X, log_input=False, reduction='sum')
                v_loss = self.lambda_v * F.poisson_nll_loss(vi_est,
self.V, log_input=False, reduction='sum')
            else:
                x_loss = self.lambda_x * F.poisson_nll_loss(xe_est,
self.X, log_input=False)
                v_loss = self.lambda_v * F.poisson_nll_loss(vi_est,
self.V, log_input=False)
            expression_loss = x_loss + v_loss
        # Regularization loss for Px, Pv, Q
        l2_reg_loss = self.l2_w * sum(torch.norm(t, p=2) ** 2 for t in
[self.Q, self.Px, self.scale_factor_visium, self.scale_factor_xenium,
self.platform_sf])
        entropy_loss = 0
        if self.add_entropy:
            if self.adaptive_entropy:
                if num_epoch % 100 == 0:
                    self.entropy_w =
np.exp(num_epoch/self.adaptive_entropy_denominator)
                entropy_loss = (self.entropy_w)  *
torch.mean(torch.sum(self.Px_soft * torch.log(self.Px_soft + 1e-12),
dim=1))
            else:
                entropy_loss = (self.entropy_w)  *
torch.mean(torch.sum(self.Px_soft * torch.log(self.Px_soft + 1e-12),
dim=1))

        l2_reg_loss -= entropy_loss

        total_loss = expression_loss + l2_reg_loss

        if verbose:
            # --- THIS LINE IS ALSO FIXED (uses dcn) ---
            term_members = [dcn(x_loss), dcn(v_loss),
dcn(l2_reg_loss)]
            term_names = ['x_loss', 'v_loss', 'l2_reg_loss']
            score_dict = dict(zip(term_names, term_members))
            clean_dict = {
                    k: score_dict[k] for k in score_dict if not
np.isnan(score_dict[k])
                }
            msg = []
            for k in clean_dict:
                m = "{}: {:.3f}".format(k, clean_dict[k])
                msg.append(m)

            print(str(msg).replace("[", "").replace("]",
```

```
"").replace("'", ""))
        return (
            total_loss,
            x_loss,
            v_loss,
            l2_reg_loss,
            entropy_loss
        )

    def train(self, num_epochs=1000, lr=1e-
3,verbose=False,loss_type='cosine',print_freq=100,warm_restart=False,
callback = None):
        print(f"Training the model with {loss_type} loss")
        train_tensors = [self.Px, self.Q, self.scale_factor_xenium,
self.scale_factor_visium, self.platform_sf]
        optimizer = torch.optim.Adam(train_tensors, lr=lr)
        if warm_restart:
            scheduler = CosineAnnealingWarmRestarts(optimizer,
T_0=1000, T_mult=2)
        keys = [
            "total_loss",
            "main_loss",
            "vg_reg",
            "kl_reg",
            "entropy_reg",
            "count_reg",
            "lambda_f_reg",
        ]
        values = [[] for i in range(len(keys))]
        training_history = {key: value for key, value in zip(keys,
values)}

        losses = []
        for epoch in range(num_epochs):
            optimizer.zero_grad()
            if verbose and epoch % 100 == 0:
                run_loss =
self.loss(verbose=True,loss_type=loss_type,num_epoch=epoch)
            else:
                run_loss = self.loss(loss_type=loss_type,
num_epoch=epoch)

            run_loss[0].backward()
            optimizer.step()

            if warm_restart:
                scheduler.step(epoch)
            if epoch % print_freq == 0:
                if loss_type == 'cosine':
                    print(f"Epoch {epoch}, Loss: {run_loss[0].item()},
```

```
Xenium cosine similarity: {2 - run_loss[1].item()/self.lambda_x},
Visium cosine similarity: {2 - run_loss[2].item()/self.lambda_v},
Regularization {run_loss[3].item()}")
                else:
                    print(f"Epoch {epoch}, Loss: {run_loss[0].item()},
Xenium (KL/Fro) divergence: {run_loss[1].item()/self.lambda_x}, Visium
(KL/Fro) divergence: {run_loss[2].item()/self.lambda_v},
Regularization {run_loss[3].item()}, Entropy {run_loss[4].item() if
self.add_entropy else 0}")
            losses.append(run_loss[0].item())
            if callback is not None:
                callback(self, epoch, run_loss)

        # Return items of interest
        with torch.no_grad():
            return {
                # --- THESE LINES ARE ALSO FIXED (uses dcn) ---
                "Px": dcn(self.Px_soft),
                "Q": dcn(self.Q_soft),
                "scale_factor_xenium": dcn(self.scale_factor_xenium),
                "scale_factor_visium": dcn(self.scale_factor_visium),
                "training_history": training_history,
                "losses": losses
            }


    def save(self, filename):
        # Save the model parameters in a config file
        # This is not a PyTorch model, but a simple class
        # with some parameters
        # Don't use torch.save for this
        # Use pickle
        import pickle
        with open(filename, "wb") as f:
            pickle.dump(self.__dict__, f)

    # load the model
    @staticmethod
    def load(filename):
        # Load the model parameters from a config file
        # This is not a PyTorch model, but a simple class
        # with some parameters
        # Don't use torch.load for this
        # Use pickle
        import pickle
        with open(filename, "rb") as f:
            obj = LowDimWithScaling.__new__(LowDimWithScaling)
            obj.__dict__.update(pickle.load(f))
        return obj
```

```
import torch
import time
import numpy as np
from anndata import AnnData

def prepare_data(vi_, xe_, holdouts, R):
    vi, xe = vi_.copy(), xe_.copy()
    vi.obsm['spatial'] = batch_apply(R, vi.obsm['spatial'])
    gamma = gen_gamma(vi.obsm['spatial'], xe.obsm['spatial'])
    vcols = set(vi.var_names)
    xcols = set(xe.var_names)
    common_genes = vcols.intersection(xcols)
    hcols = set(holdouts).intersection(common_genes)
    train_genes = common_genes - hcols
    test_genes = hcols - train_genes
    sorted_xe = xe[:, list(train_genes)].copy()
    vi_genes = list(train_genes) + list(test_genes)
    sorted_vi = vi[:, vi_genes]
    return sorted_vi, sorted_xe, gamma


def impute_work(vi, xe, gamma, hdim, num_epoch, plsf, seed,
ent_offset, ent_div, device="cuda:0"):

    torch.manual_seed(seed)
    np.random.seed(seed)
    model = LowDimWithScaling(xe, vi, gamma, n_factors = hdim, device
= device,
                                        adaptive_entropy = True,
adaptive_entropy_denominator = ent_div,
                                        poisson_sum = True, add_entropy
= True, platform_scaling = plsf)
    _ = model.train(num_epochs = num_epoch + 1, lr=5e-2,
print_freq=1000, loss_type='poisson')
    full_pred = AnnData(X = dcn(model.get_xenium_est()), obs = xe.obs,
var = vi.var)
    return [], full_pred.copy()  # Return empty list instead of
undefined cb.data


def impute_genes(vi, xe, holdouts, hdim, R = None, ent_div = 1000,
seed = 10042, lr = 1e-3, epochs = 5000, device = "cuda:0"):
    t = time.time()
    if R is None:
        R = np.eye(3)
    sorted_vi, sorted_xe, gamma = prepare_data(vi, xe, holdouts, R)
    torch.manual_seed(seed)
    np.random.seed(seed)
    model = LowDimWithScaling(sorted_xe,
                                        sorted_vi,
```

```
                                        gamma,
                                        n_factors = hdim,
                                        device = device,
                                        adaptive_entropy = True,
                                        adaptive_entropy_denominator =
ent_div,
                                        poisson_sum = True,
                                        add_entropy = True,
                                        platform_scaling = True)
    _ = model.train(num_epochs = epochs + 1, lr=lr, print_freq = 500,
loss_type='poisson')
    full_pred = AnnData(X = dcn(model.get_xenium_est()), obs =
sorted_xe.obs, var = sorted_vi.var)
    print("Training finished in (seconds):", time.time() - t)
    return full_pred[:, list(x for x in holdouts if x in
full_pred.var_names)].copy()

def infer_latent_types(vi, xe, hdim, R = None, ent_div = 400, seed =
10042, lr = 1e-3, epochs = 5000, device = "cuda:0"):
    t = time.time()
    if R is None:
        R = np.eye(3)
    sorted_vi, sorted_xe, gamma = prepare_data(vi, xe, [], R)
    torch.manual_seed(seed)
    np.random.seed(seed)
    model = LowDimWithScaling(sorted_xe,
                                        sorted_vi,
                                        gamma,
                                        n_factors = hdim,
                                        device = device,
                                        adaptive_entropy = True,
                                        adaptive_entropy_denominator =
ent_div,
                                        poisson_sum = True,
                                        add_entropy = True,
                                        platform_scaling = True)
    _ = model.train(num_epochs = epochs + 1, lr=lr, print_freq = 500,
loss_type='poisson')
    full_pred = AnnData(X = dcn(model.get_xenium_est()), obs =
sorted_xe.obs, var = sorted_vi.var)
    print("Training finished in (seconds):", time.time() - t)
    ret_x = dcn(model.Px_soft)
    ret_v = dcn(torch.mm(model.gamma.T,
torch.exp(model.scale_factor_visium) * model.Px_soft))
    ret_v = ret_v / (1e-10 + np.sum(ret_v, axis = 1, keepdims = True))
    Px = AnnData(X = ret_x, obs = sorted_xe.obs, obsm =
sorted_xe.obsm)
    Pv = AnnData(X = ret_v, obs = sorted_vi.obs, obsm =
```

```
sorted_vi.obsm)
    return Px, Pv

import sys
import os.path
# All lib files are in the same directory, no need for complex path
manipulation

holdouts = list(xe.var_names)[::10]
# Use CPU instead of CUDA since CUDA is not available
ret = impute_genes(vi, xe, holdouts, 20, R, lr = 1e-2, epochs = 5000,
device="cuda")
```

```
/tmp/ipykernel_48/473258281.py:22: UserWarning: Sparse CSR tensor
support is in beta state. If you miss a functionality in the sparse
tensor support, please submit a feature request to
https://github.com/pytorch/pytorch/issues. (Triggered internally at
/pytorch/aten/src/ATen/SparseCsrTensorImpl.cpp:53.)
  sparse_csr_tensor = torch.sparse_csr_tensor(crow_indices,
col_indices, values, size=array.shape)

Training the model with poisson loss
Epoch 0, Loss: 134590688.0, Xenium (KL/Fro) divergence: 127679984.0,
Visium (KL/Fro) divergence: 6910701.0, Regularization
6.354728698730469, Entropy -2.9957265853881836
Epoch 500, Loss: -3459679.75, Xenium (KL/Fro) divergence:
271422.28125, Visium (KL/Fro) divergence: -3731354.75, Regularization
252.8483123779297, Entropy -2.025367498397827
Epoch 1000, Loss: -5755972.0, Xenium (KL/Fro) divergence: -1798326.25,
Visium (KL/Fro) divergence: -3957980.0, Regularization
334.0763244628906, Entropy -3.580296754837036
Epoch 1500, Loss: -5993926.0, Xenium (KL/Fro) divergence: -2012244.0,
Visium (KL/Fro) divergence: -3982077.5, Regularization
395.60516357421875, Entropy -6.162074089050293
Epoch 2000, Loss: -6132309.0, Xenium (KL/Fro) divergence: -2142224.5,
Visium (KL/Fro) divergence: -3990535.0, Regularization
450.7391662597656, Entropy -10.540770530700684
Epoch 2500, Loss: -6214156.0, Xenium (KL/Fro) divergence: -2220476.0,
Visium (KL/Fro) divergence: -3994184.0, Regularization
503.8758850097656, Entropy -17.844030380249023
Epoch 3000, Loss: -6268764.0, Xenium (KL/Fro) divergence: -2272976.75,
Visium (KL/Fro) divergence: -3996346.25, Regularization
559.0385131835938, Entropy -29.867185592651367
Epoch 3500, Loss: -6307613.5, Xenium (KL/Fro) divergence: -2310015.5,
Visium (KL/Fro) divergence: -3998219.75, Regularization
621.3392944335938, Entropy -49.381309509277344
Epoch 4000, Loss: -6328117.0, Xenium (KL/Fro) divergence: -2328429.75,
Visium (KL/Fro) divergence: -4000382.5, Regularization
695.1068725585938, Entropy -81.92835235595703
Epoch 4500, Loss: -6342741.5, Xenium (KL/Fro) divergence: -2341958.5,
```

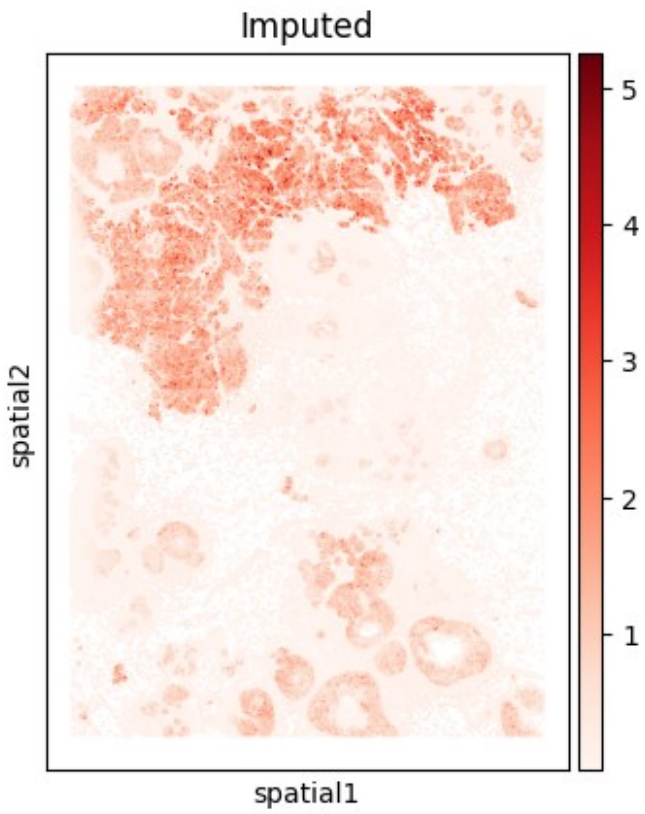
```
Visium (KL/Fro) divergence: -4001570.5, Regularization
787.6446533203125, Entropy -135.9194793701172
Epoch 5000, Loss: -6355542.5, Xenium (KL/Fro) divergence: -2354032.5,
Visium (KL/Fro) divergence: -4002422.75, Regularization
912.3714599609375, Entropy -224.97702026367188
Training finished in (seconds): 119.11488699913025
```

```
print('average holdout R^2:', avg_corr_by_col(ret.X, xe[:,
ret.var_names].X))
```

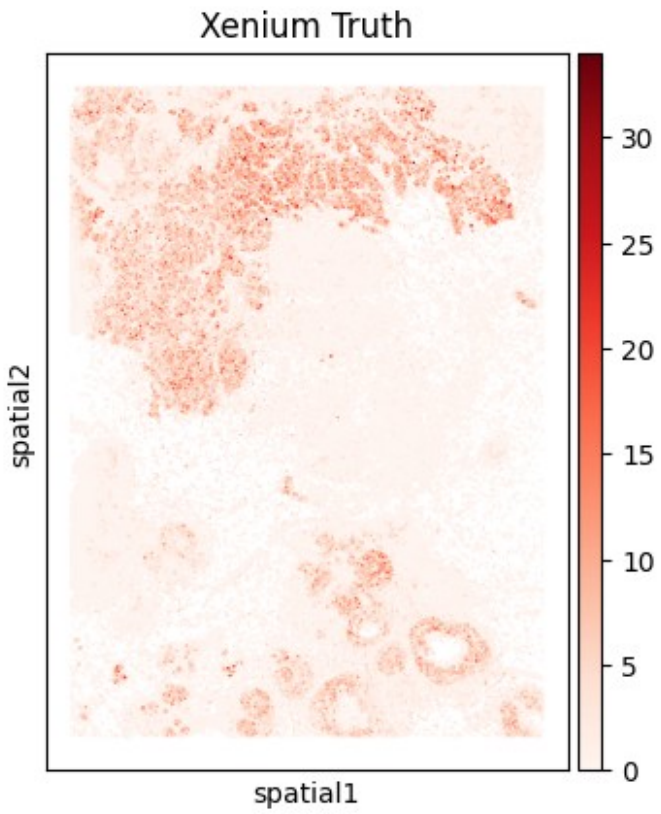
```
average holdout R^2: 0.25849962603430043
```

```
import scanpy as sc
ret.obsm['spatial'] = xe.obsm['spatial']
g = holdouts[0]
sc.pl.spatial(ret, color = [g], spot_size=30, title = "Imputed",
color_map = "Reds")

# The non-breaking space before color_map has been replaced with a
regular space:
sc.pl.spatial(xe, color = [g], spot_size=30, title = "Xenium Truth",
color_map = "Reds")
```

```
/tmp/ipykernel_48/198286621.py:4: FutureWarning: Use
`squidpy.pl.spatial_scatter` instead.
  sc.pl.spatial(ret, color = [g], spot_size=30, title = "Imputed",
color_map = "Reds")
```




```
/tmp/ipykernel_48/198286621.py:7: FutureWarning: Use
`squidpy.pl.spatial_scatter` instead.
  sc.pl.spatial(xe, color = [g], spot_size=30, title = "Xenium Truth",
color_map = "Reds")
```




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

# Add the spatial coordinates to your 'ret' object
ret.obsm['spatial'] = xe.obsm['spatial']

# Loop over ret.var_names (the genes that were actually imputed)
print(f"Generating plots for all {len(ret.var_names)} imputed holdout
genes...")

for g in ret.var_names:

    # Calculate the R^2 score for this specific gene

    # --- THIS IS THE CORRECTED LINE ---
    # The ground truth (xe) should be the first argument,
    # and the estimate (ret) should be the second.
    gene_r2 = avg_corr_by_col(
        xe[:, [g]].X,   # Ground Truth
        ret[:, [g]].X    # Estimate
    )
    # ----------------------------------

    # # Create a figure with two plots
```

```
    # fig, axes = plt.subplots(1, 2, figsize=(12, 6))

    print(f"\nGene: {g} (R²: {gene_r2:.3f})")

#     # Plot 1: Imputed (from 'ret')
#     sc.pl.spatial(
#         ret,
#         color=[g],
#         spot_size=30,
#         title=f"Imputed: {g}",
#         color_map="Purples",
#         ax=axes[0],
#         show=False
#     )

#     # Plot 2: Xenium Truth (from 'xe')
#     sc.pl.spatial(
#         xe,
#         color=[g],
#         spot_size=30,
#         title=f"Xenium Truth: {g}",
#         color_map="Purples",
#         ax=axes[1],
#         show=False
#     )

#     # Show the combined plot for this gene
#     plt.tight_layout()
#     plt.show()

# print("\nAll plots generated.")
```

```
Generating plots for all 31 imputed holdout genes...

Gene: ABCC11 (R²: 0.471)

Gene: AQP1 (R²: 0.497)

Gene: C1QC (R²: 0.406)

Gene: CCL8 (R²: 0.012)

Gene: CD274 (R²: 0.011)

Gene: CD83 (R²: 0.006)

Gene: CENPF (R²: 0.154)

Gene: CRISPLD2 (R²: 0.441)
```

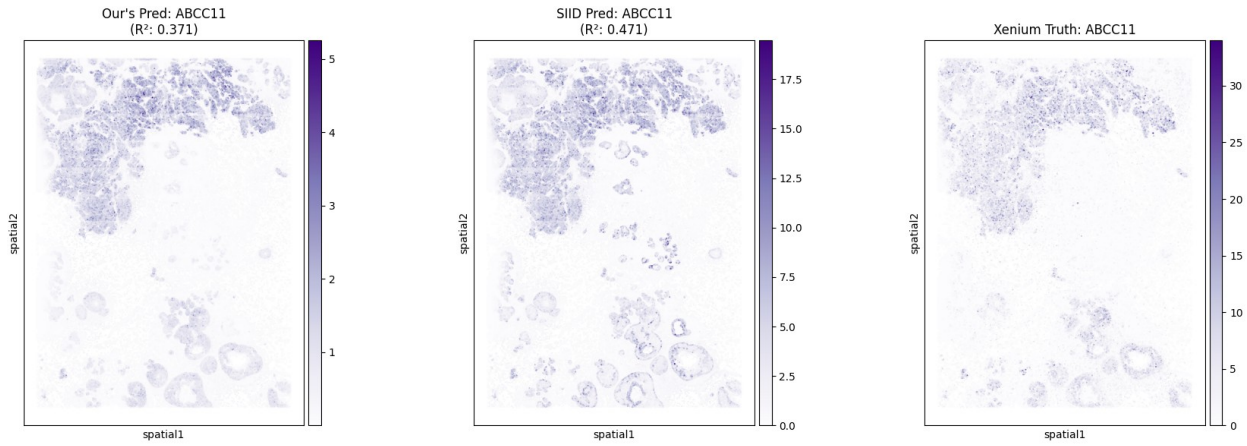
```
Gene: CXCR4 (R²: 0.431)

Gene: DSP (R²: 0.423)

Gene: ELF5 (R²: 0.032)

Gene: FBLIM1 (R²: 0.239)

Gene: FSTL3 (R²: 0.098)

Gene: HDC (R²: 0.001)

Gene: IL3RA (R²: 0.186)

Gene: KLRB1 (R²: 0.255)

Gene: KRT7 (R²: 0.561)

Gene: LPXN (R²: 0.125)

Gene: MKI67 (R²: 0.100)

Gene: MUC6 (R²: 0.002)

Gene: NOSTRIN (R²: 0.312)

Gene: PDGFRA (R²: 0.538)

Gene: POSTN (R²: 0.737)

Gene: RAMP2 (R²: 0.526)

Gene: SCD (R²: 0.306)

Gene: SFRP4 (R²: 0.293)

Gene: SOX17 (R²: 0.180)

Gene: TCEAL7 (R²: 0.099)

Gene: TIGIT (R²: 0.084)

Gene: TRAPPC3 (R²: 0.156)

Gene: ZEB1 (R²: 0.332)
```

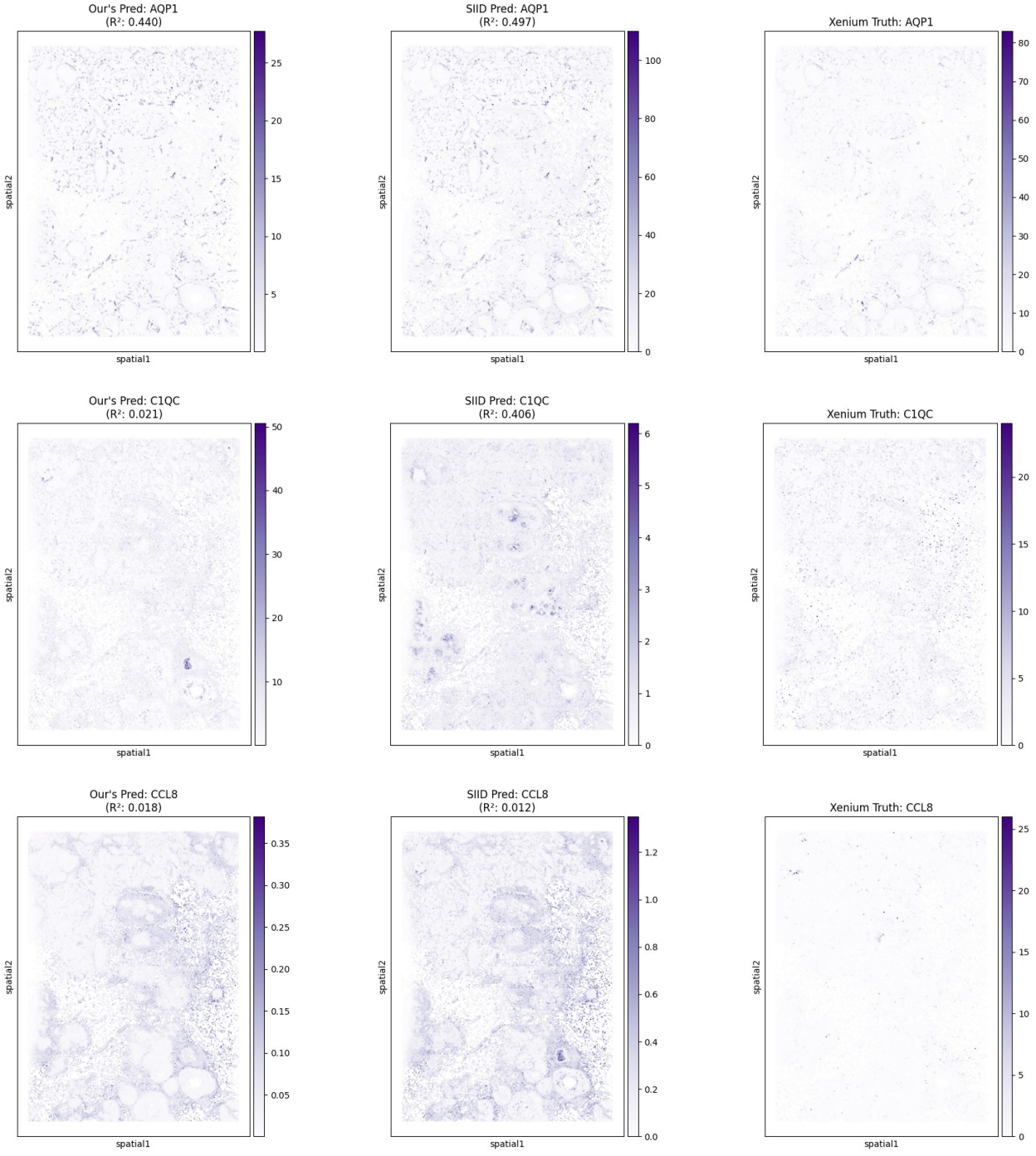
```python
import scanpy as sc
import anndata as ad
import numpy as np

imputed_matrix =
np.load('/kaggle/input/xen-imputed/best_imputed_xenium.npy')
```

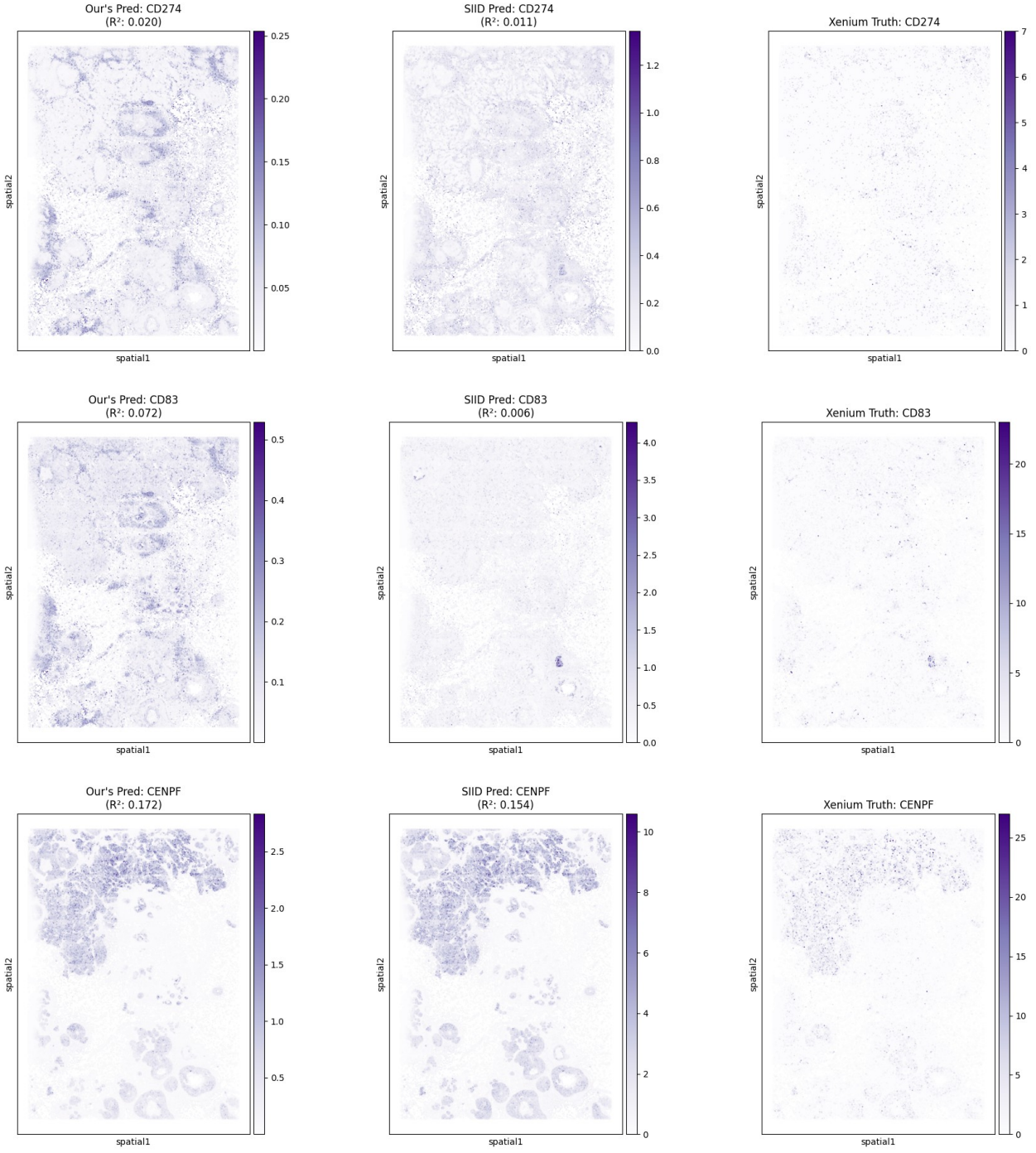
```
imputed_ad = ad.AnnData(X=imputed_matrix, obs=xe.obs, var=vi.var)

import scanpy as sc
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')


ret.obsm['spatial'] = xe.obsm['spatial']
imputed_ad.obsm['spatial'] = xe.obsm['spatial']


print(f"Generating plots for all {len(ret.var_names)} imputed holdout
genes...")

for g in ret.var_names:

    # Calculate the R^2 score for this specific gene

    # The ground truth (xe) should be the first argument,
    # and the estimate (ret) should be the second.
    gene_r2_siid = avg_corr_by_col(
        xe[:, [g]].X,   # Ground Truth
        ret[:, [g]].X    # Estimate (SIID)
    )
    gene_r2_ours = avg_corr_by_col(
        xe[:, [g]].X,   # Ground Truth
        imputed_ad[:, [g]].X   # Estimate (ours)
    )
    # ---------------------------------

    fig, axes = plt.subplots(1, 3, figsize=(18, 6))

     # Plot 3: Xenium Truth (from 'xe')
    sc.pl.spatial(
        xe,
        color=[g],
        spot_size=30,
        title=f"Xenium Truth: {g}",
        color_map="Purples",
        ax=axes[2], # Plot on the third axis
        show=False
    )

    # Plot 1: Our's Pred (using 'ret')
    sc.pl.spatial(
        ret,
        color=[g],
        spot_size=30,
        # Title now matches the R² variable 'gene_r2_ours'
```

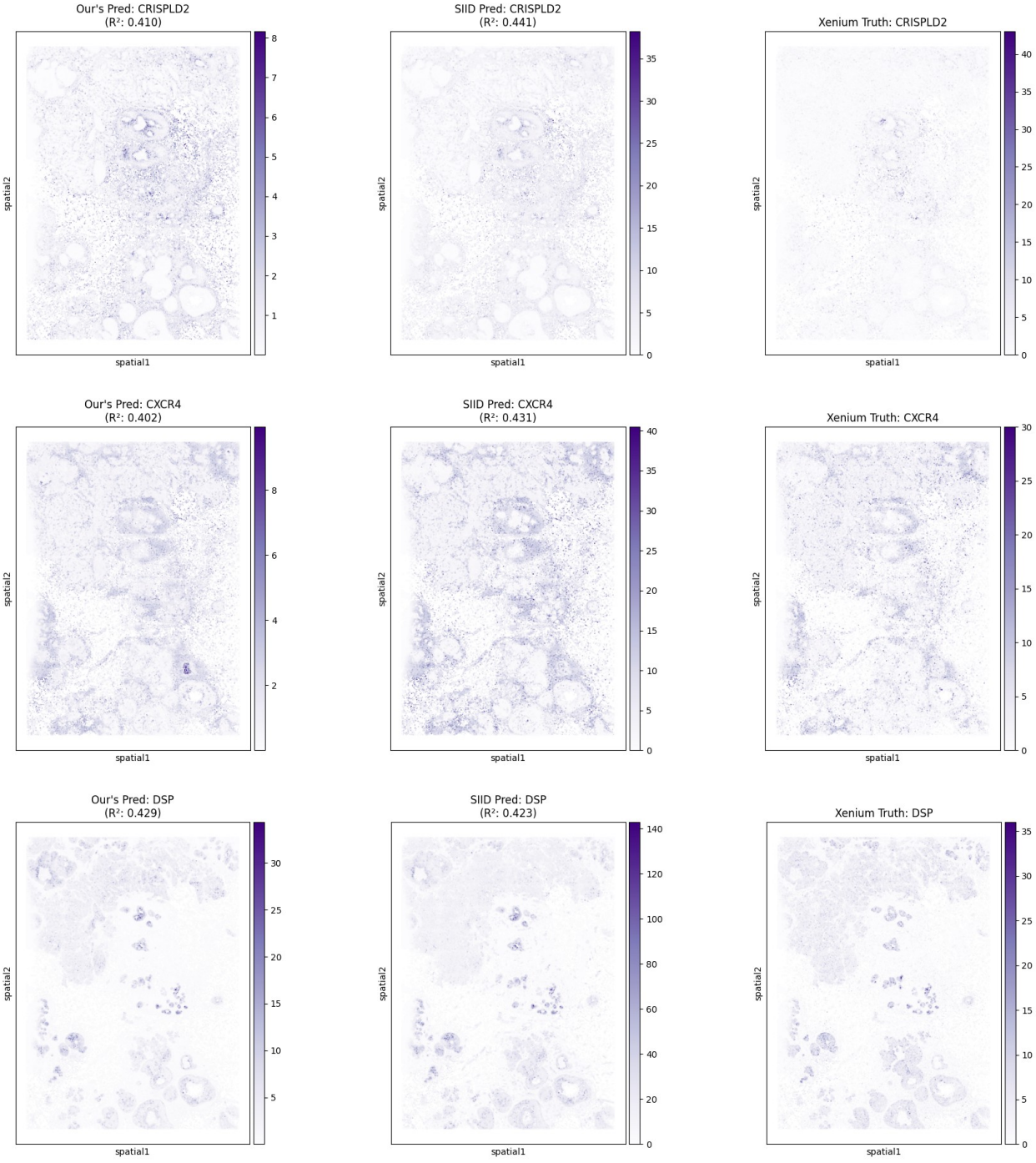
```
        title=f"Our's Pred: {g}\n(R²: {gene_r2_ours:.3f})",
        color_map="Purples",
        ax=axes[0], # Plot on the first axis
        show=False
    )

    # Plot 2: SIID Pred (using 'imputed_ad')
    sc.pl.spatial(
        imputed_ad,
        color=[g],
        spot_size=30,
        # Title now matches the R² variable 'gene_r2_siid'
        title=f"SIID Pred: {g}\n(R²: {gene_r2_siid:.3f})",
        color_map="Purples",
        ax=axes[1], # Plot on the second axis
        show=False
    )



    # Show the combined plot for this gene
    plt.tight_layout()
    plt.show()

print("\nAll plots generated.")
```

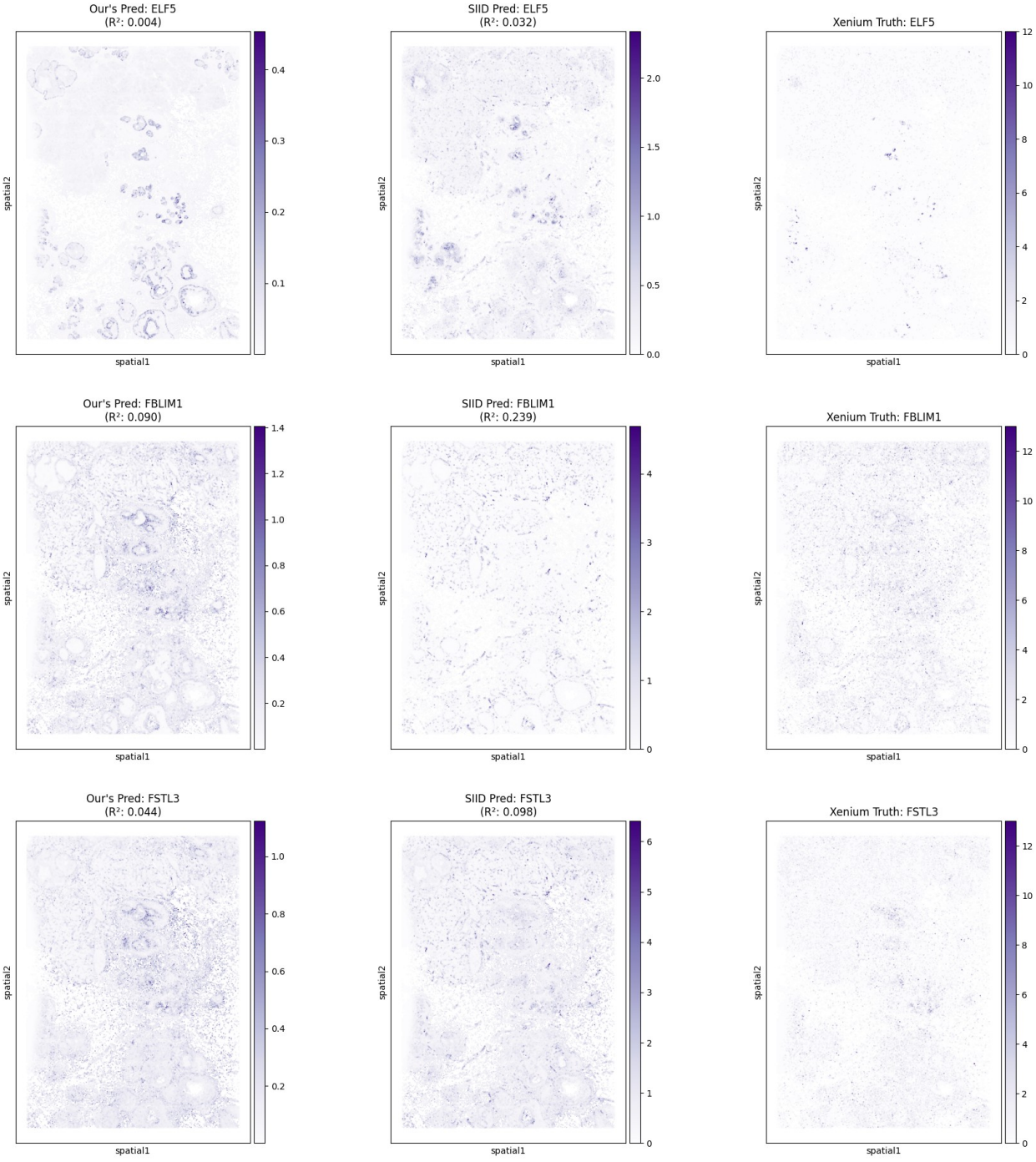
```
Generating plots for all 31 imputed holdout genes...
```

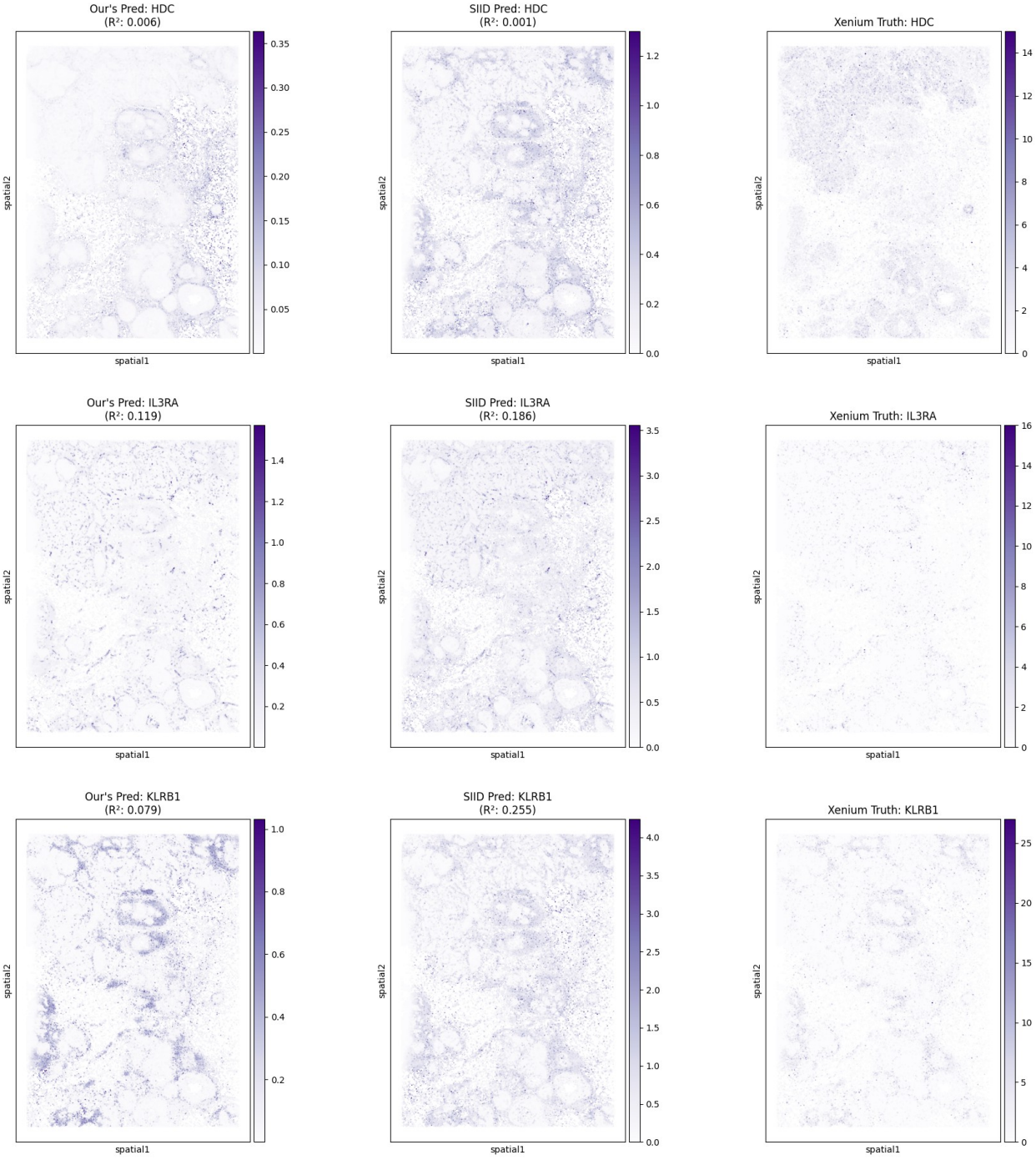



Our's Pred: AQP1
(R²: 0.440)
SIID Pred: AQP1
(R²: 0.497)
Xenium Truth: AQP1
Our's Pred: C1QC
(R²: 0.021)
SIID Pred: C1QC
(R²: 0.406)
Xenium Truth: C1QC
Our's Pred: CCL8
(R²: 0.018)
SIID Pred: CCL8
(R²: 0.012)
Xenium Truth: CCL8
spatial1
spatial2

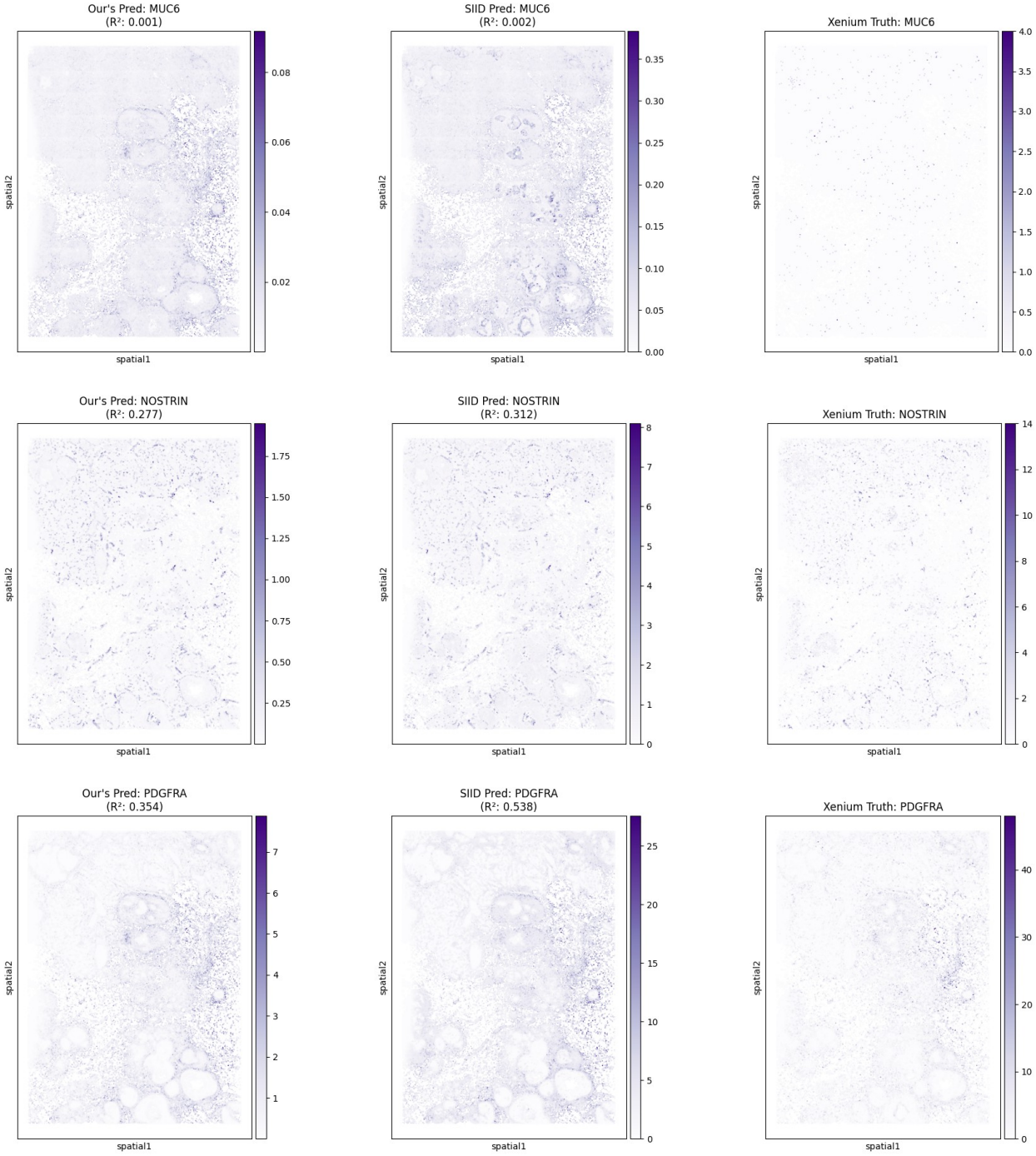

Our's Pred: CD274
(R²: 0.020)
SIID Pred: CD274
(R²: 0.011)
Xenium Truth: CD274
Our's Pred: CD83
(R²: 0.072)
SIID Pred: CD83
(R²: 0.006)
Xenium Truth: CD83
Our's Pred: CENPF
(R²: 0.172)
SIID Pred: CENPF
(R²: 0.154)
Xenium Truth: CENPF
spatial1
spatial2

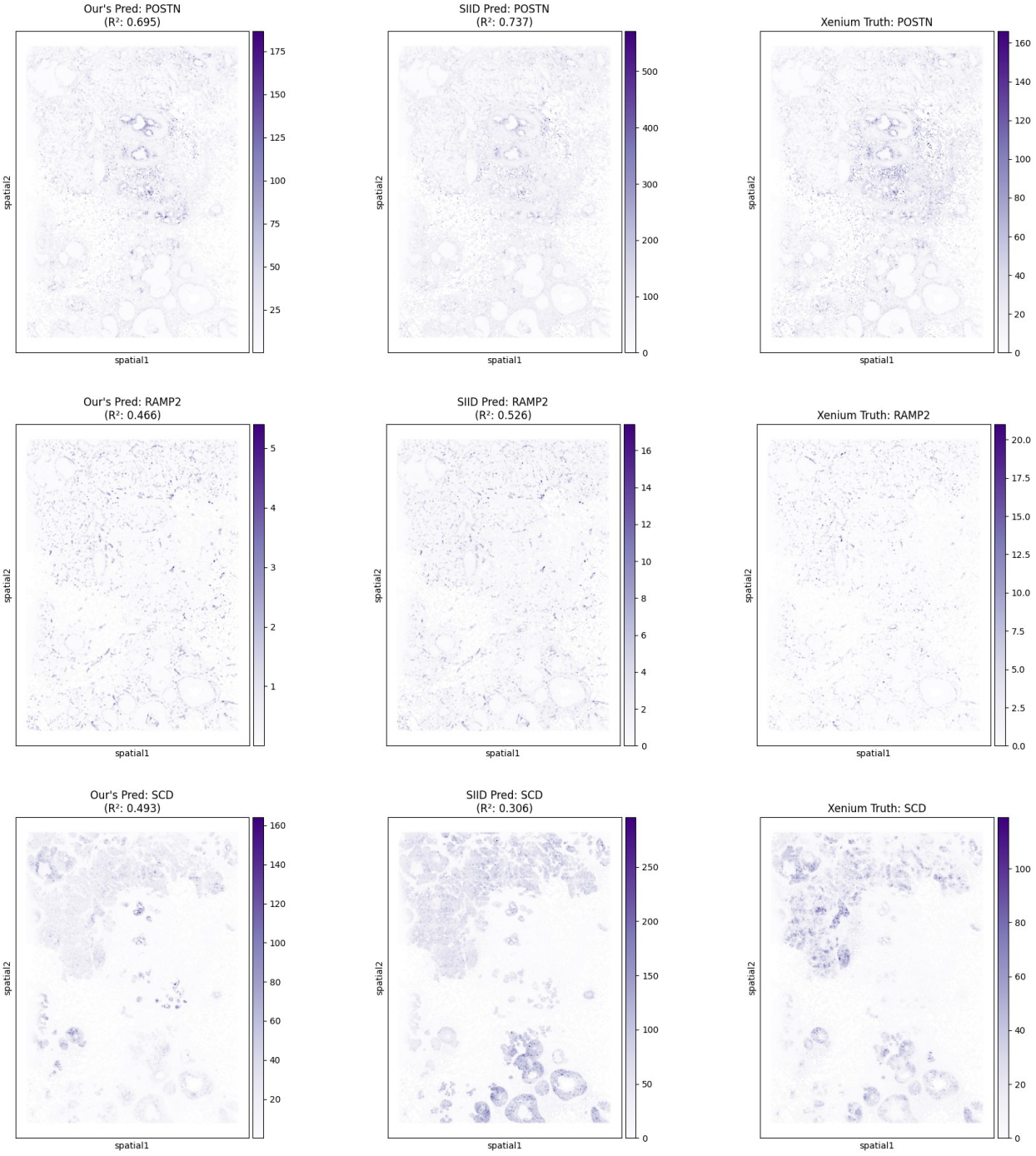

Our's Pred: CRISPLD2
(R²: 0.410)
SIID Pred: CRISPLD2
(R²: 0.441)
Xenium Truth: CRISPLD2
Our's Pred: CXCR4
(R²: 0.402)
SIID Pred: CXCR4
(R²: 0.431)
Xenium Truth: CXCR4
Our's Pred: DSP
(R²: 0.429)
SIID Pred: DSP
(R²: 0.423)
Xenium Truth: DSP
spatial1
spatial2

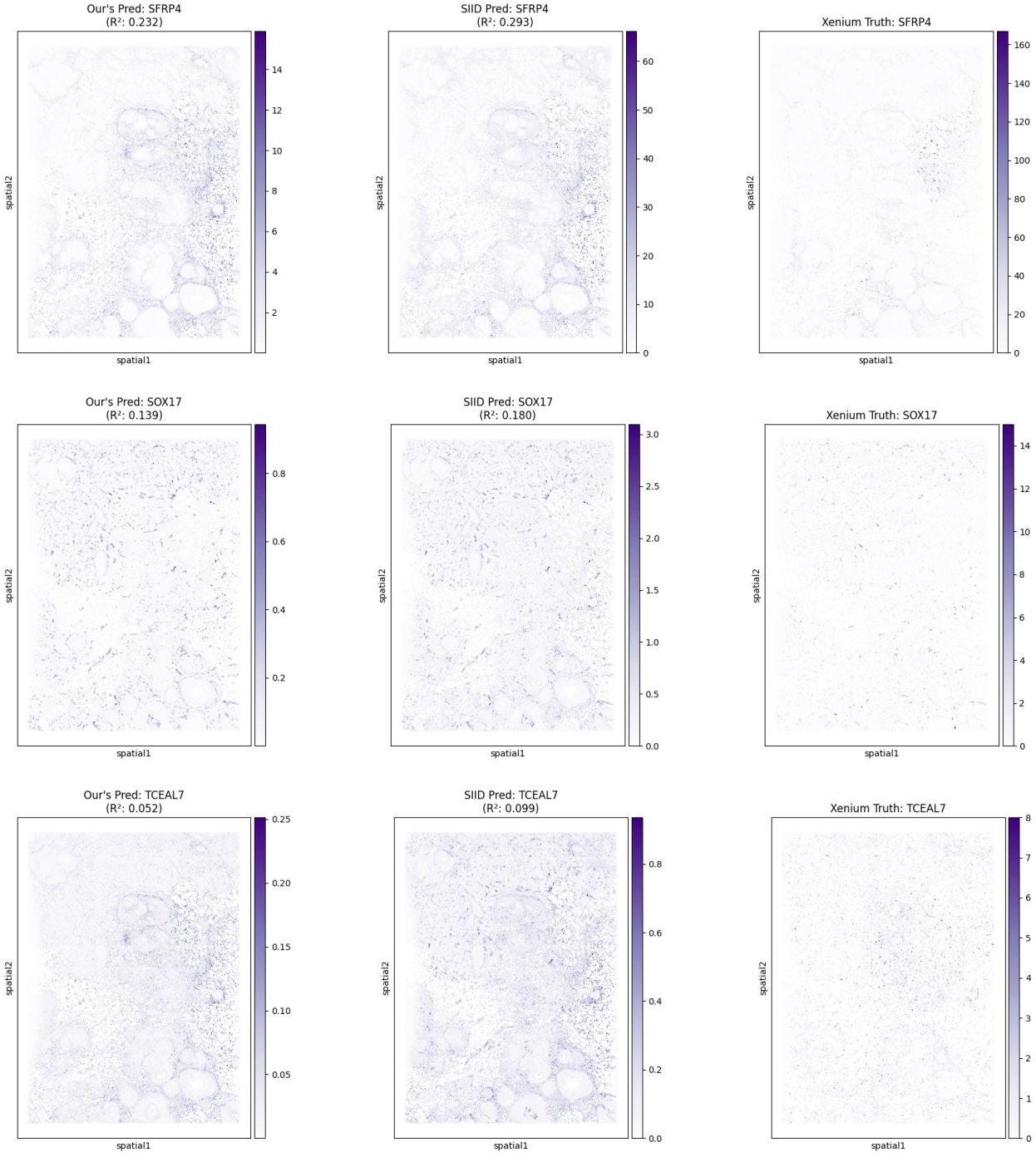

Our's Pred: ELF5
(R²: 0.004)
SIID Pred: ELF5
(R²: 0.032)
Xenium Truth: ELF5
Our's Pred: FBLIM1
(R²: 0.090)
SIID Pred: FBLIM1
(R²: 0.239)
Xenium Truth: FBLIM1
Our's Pred: FSTL3
(R²: 0.044)
SIID Pred: FSTL3
(R²: 0.098)
Xenium Truth: FSTL3
spatial1
spatial2

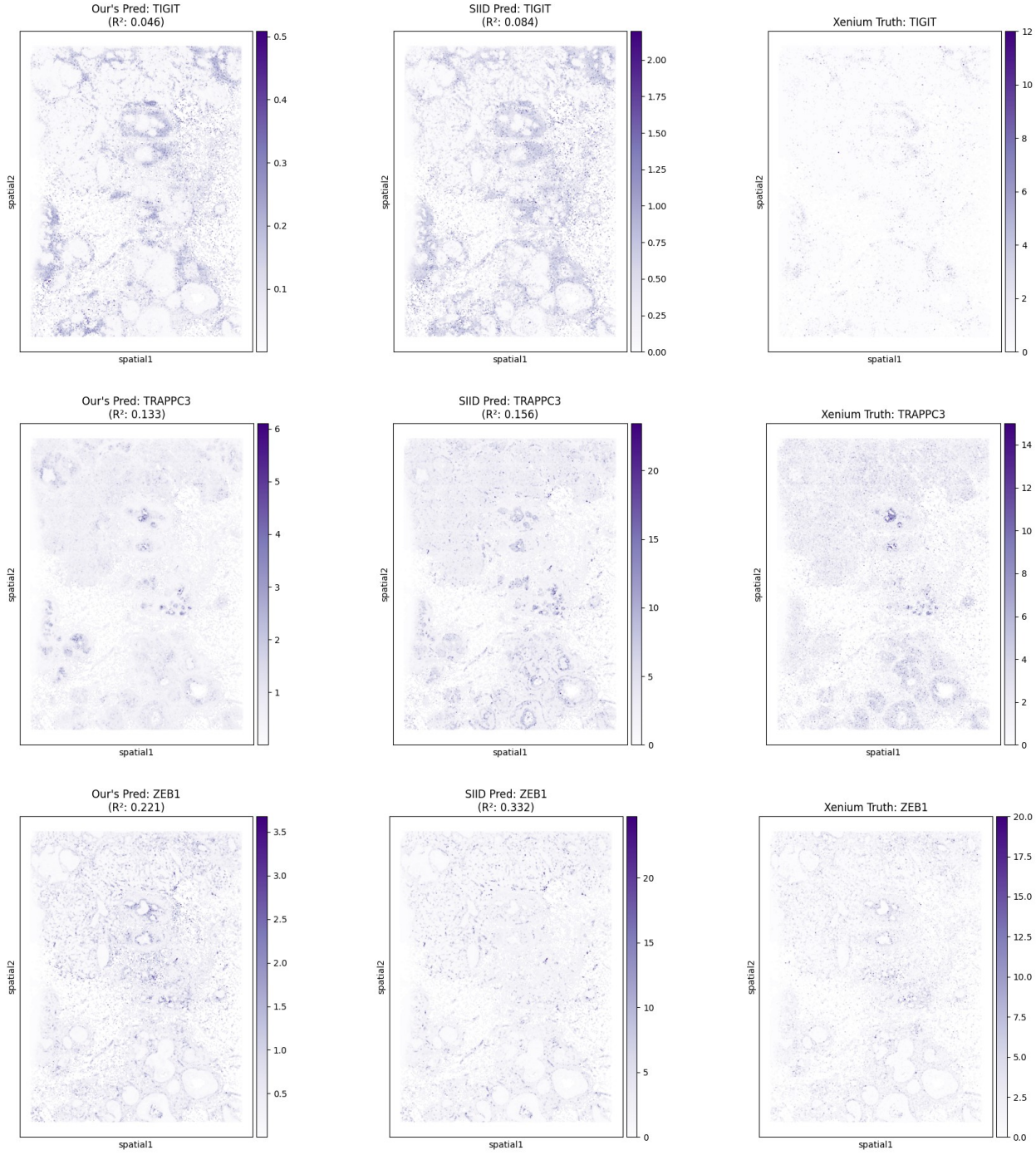

Our's Pred: HDC
(R²: 0.006)
SIID Pred: HDC
(R²: 0.001)
Xenium Truth: HDC
Our's Pred: IL3RA
(R²: 0.119)
SIID Pred: IL3RA
(R²: 0.186)
Xenium Truth: IL3RA
Our's Pred: KLRB1
(R²: 0.079)
SIID Pred: KLRB1
(R²: 0.255)
Xenium Truth: KLRB1
spatial1
spatial2


Our's Pred: KRT7
(R²: 0.757)
SIID Pred: KRT7
(R²: 0.561)
Xenium Truth: KRT7
Our's Pred: LPXN
(R²: 0.182)
SIID Pred: LPXN
(R²: 0.125)
Xenium Truth: LPXN
Our's Pred: MKI67
(R²: 0.109)
SIID Pred: MKI67
(R²: 0.100)
Xenium Truth: MKI67
spatial1
spatial2


Our's Pred: MUC6
(R²: 0.001)
SIID Pred: MUC6
(R²: 0.002)
Xenium Truth: MUC6
Our's Pred: NOSTRIN
(R²: 0.277)
SIID Pred: NOSTRIN
(R²: 0.312)
Xenium Truth: NOSTRIN
Our's Pred: PDGFRA
(R²: 0.354)
SIID Pred: PDGFRA
(R²: 0.538)
Xenium Truth: PDGFRA
spatial1
spatial2


Our's Pred: POSTN
(R²: 0.695)
SIID Pred: POSTN
(R²: 0.737)
Xenium Truth: POSTN
Our's Pred: RAMP2
(R²: 0.466)
SIID Pred: RAMP2
(R²: 0.526)
Xenium Truth: RAMP2
Our's Pred: SCD
(R²: 0.493)
SIID Pred: SCD
(R²: 0.306)
Xenium Truth: SCD
spatial1
spatial2


Our's Pred: SFRP4
(R²: 0.232)
SIID Pred: SFRP4
(R²: 0.293)
Xenium Truth: SFRP4
Our's Pred: SOX17
(R²: 0.139)
SIID Pred: SOX17
(R²: 0.180)
Xenium Truth: SOX17
Our's Pred: TCEAL7
(R²: 0.052)
SIID Pred: TCEAL7
(R²: 0.099)
Xenium Truth: TCEAL7
spatial1
spatial2




```
All plots generated.
```