

RASP Supplemental Information

Ian K. Gingerich^{1,2,*}, Brittany A. Goods², and H. Robert Frost^{1,*}

¹Department of Biomedical Data Science, Geisel School of Medicine, Dartmouth College, Hanover, NH, USA

²Thayer School of Engineering, Dartmouth College, Hanover, NH, USA

*Corresponding authors: ian.gingerich.gr@dartmouth.edu, rob.frost@dartmouth.edu

Contents

1	Supplemental Methods	2
1.1	Efficient numerical methods and data structures utilized by RASP	2
1.2	Adjusted Rand Index calculation	4
1.3	CHAOS score calculation	6
1.4	Moran's I calculation	6
1.5	Real world data preprocessing	6
1.6	Simulated dataset processing	9
2	Supplemental Results	12
2.1	A note on Moran's I and CHAOS score	12

List of Figures

1	Extended Data Fig.1	13
2	Extended Data Fig.2	14
3	Extended Data Fig.3	15
4	Extended Data Fig. 4	16
5	Extended Data Fig. 5	17
6	Extended Data Fig.6	18
7	Extended Data Fig.7	19
8	Extended Data Fig.8	20
9	Extended Data Fig.9	21
10	Extended Data Fig.10	22
11	Extended Data Fig.11	23
12	Extended Data Fig.12	24
13	Extended Data Fig.13	25
14	Extended Data Fig.14	26
15	Extended Data Fig.15	27
16	Extended Data Fig.16	28
17	Extended Data Fig.17	29
18	Extended Data Fig.18	30

List of Algorithms

1	Randomized SVD	3
2	KD-Tree Construction	4
3	Build distance matrix from KD-Tree	5

39	4	query_ball_point	5
40	5	local_density	6
41	6	CHAOS Function	7
42	7	1-Nearest Neighbor Function	7
43	8	Moran's I Calculation	8

44 List of Tables

45	1	Datasets used for RASP evaluation	9
46	2	Parameter Selection	9

47 1 Supplemental Methods

48 1.1 Efficient numerical methods and data structures utilized by RASP

49 Randomized PCA/SVD

50 The RASP algorithm depends on randomized principal component analysis (rPCA)[1] to perform computationally
51 efficient dimensionality reduction on the gene expression data. For the **Python** version of RASP, this is imple-
52 mented using the **sklearn** (version 1.5.2) package[2], which has an option for randomized PCA when calling the
53 **PCA()** function. For the **R** version of RASP, this is implemented using the **rpca** function in the **rsvd** package
54 (version 1.0.5) [1]. This randomized approach builds on the impressive results from the field of randomized numer-
55 ical linear algebra (RNLA)[3] and allows an approximate reduced rank embedding to be computed for very large
56 datasets at a computational cost that is orders-of-magnitude faster than can be achieved using non-stochastic trun-
57 cated PCA implementations. Randomized PCA is implemented using a randomized singular value decomposition
58 (rSVD) , which includes the following key steps (see Algorithm 1 for a detailed description):

- 59 • **Random sampling:** Generate a random projection matrix that reduces the dimensions of the original data
60 while preserving its structure.
- 61 • **Compute a non-stochastic SVD on the random embedding:** Perform a standard deterministic SVD
62 on this lower-dimensional representation of the data.
- 63 • **Generate an approximate SVD for the full matrix:** The results of the smaller SVD are used to
64 generate an approximate truncated SVD for the original matrix.

65 K-dimensional tree (KD-Tree) operations

66 The RASP method also relies on the use of the KD-Tree data structure for organizing spatial coordinates in k -
67 dimensional space for fast range or nearest neighbor searches. Construction of the KD-Tree structure is detailed in
68 Algorithm 2. For the **Python** version of RASP, this algorithm is implemented using the **KDTree** function provided
69 by the **Scipy** package [4] (version 1.14.1). For the **R** version of RASP, this algorithm is implemented using the
70 **nn2** function from the **RANN** package (version 2.6.2).

71 The KD-Tree structure can be described at a high-level as follows:

- 72 • **Node Structure:** Each node N contains:
 - 73 – A point $p \in \mathbf{R}^k$: representing the location in k -dimensional space.
 - 74 – Left child N_{left} and right child N_{right} : representing subtrees corresponding to the divided space.
- 75 • **Partitioning:** The space is partitioned based on the median point along the chosen dimension d (where d
76 cycles through dimensions $1, 2, \dots, k$):
 - 77 – If the current dimension is d , nodes are divided into left and right subtrees based on whether their
78 points have coordinates less than or greater than the median along dimension d .

Algorithm 1 Randomized SVD

Inputs:

- Matrix A (size $m \times n$): the input data matrix.
- Integer k : the number of significant singular values to retain.
- Integer p : oversampling parameter (to improve accuracy).

Outputs:

- Matrix U_k (size $m \times k$): left singular vectors.
- Matrix S_k (size $k \times k$): diagonal matrix of singular values.
- Matrix V_k (size $n \times k$): right singular vectors.

Notation:

- Matrix Ω (size $n \times r$): random projection matrix.
- Matrix Y (size $m \times r$): projected matrix $A\Omega$.
- Matrices Q and R : orthonormal basis and upper triangular matrix from QR decomposition.
- Matrix B (size $r \times n$): matrix after projection onto Q .

```
1:  $r = k + p$                                 ▷ Dimension of the random projection
2: Generate a random matrix  $\Omega$  (size  $n \times r$ )  ▷ Entries sampled from a Gaussian or uniform
   distribution over a specified range.
3:  $Y = A \cdot \Omega$                             ▷ Project original matrix
4:  $[Q, R] \leftarrow \text{QR}(Y)$                     ▷ QR decomposition
5:  $B = Q^T \cdot A$                                 ▷ Form smaller matrix
6:  $[U_b, S_b, V_b] \leftarrow \text{SVD}(B)$           ▷ Compute SVD of  $B$ 
7:  $U_k = Q \cdot U_b$                             ▷ Form left singular vectors
8:  $S_k = S_b[1:k, 1:k]$                         ▷ Retain top  $k$  singular values
9:  $V_k = V_b[1:k, :]$                             ▷ Retain top  $k$  right singular vectors
   return  $U_k, S_k, V_k$ 
```

- **Recursive Construction:** This process continues recursively for each subtree until a specified stopping condition is met (e.g., a maximum number of points in a leaf node).

Algorithm 2 KD-Tree Construction

Inputs:

- Set of points P : a list of points in $\mathbf{R}^k$
- Integer $depth$: current depth in the tree (or dimension to split on)

Outputs:

- Node N : root of the constructed KD-Tree

```

1: if size of  $P$  is 0 then return None           ▷ Base case: no points to construct a node
2:  $k \leftarrow$  number of dimensions
3:  $axis \leftarrow depth \bmod k$                  ▷ Choose axis for this level
4: Sort points  $P$  along the chosen axis
5:  $median \leftarrow \lfloor \text{size}(P)/2 \rfloor$            ▷ Find median index
6:  $N \leftarrow$  new node  $p_{median}$                ▷ Create a node at the median
7:  $N.left \leftarrow \text{KD-Tree}(P[0 : median], depth + 1)$ 
8:  $N.right \leftarrow \text{KD-Tree}(P[median + 1 : end], depth + 1)$ 
   return  $N$                                    ▷ Return the constructed node

```

The KD-Tree is used to calculate the sparse distance matrix leveraged for spatial smoothing. The KD-Tree can be queried to find distances to all neighbors within a specified distance, or find all distances to K-nearest-neighbors, see `Build distance matrix from KD-Tree3`.

The KD-Tree is also used when performing local cell density calculations. The **SciPy** function `query_ball_point()` is used to identify all points that fall within a specified radius r of a given point. The `nn2` function is used for this purpose in the **R** version of RASP. These functions exploit the KD-Tree's partitioning to quickly find neighboring points without needing to check every point in the dataset. For each point, the density is computed as the number of neighboring points found divided by the volume of the neighborhood. In a two-dimensional space, the volume is calculated using the area of a circle with radius r (i.e., πr^2). This formula ensures that the density reflects the spatial distribution of points around each queried point. See `local_density()`⁵ and `query_ball_point()`⁴.

1.2 Adjusted Rand Index calculation

For the ovary, breast cancer, DLPFC, and simulated datasets we compared the identified spatial domains or cell type annotations directly to the ground truth labels using the adjusted rand index (ARI)[5] by using the `adjusted_rand_score` function from the **sklearn** package (version 1.5.2) and `adjustedRandIndex` from the **mclust** library (6.1.1) in the **R** version of RASP. Mathematically, the ARI is defined as:

$$ARI = \frac{\sum_{ij} \binom{n_{ij}}{2} - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2} \right] / \binom{n}{2}}{\frac{1}{2} \left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2} \right] - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2} \right] / \binom{n}{2}}$$

Where:

- n is the total number of elements (e.g., cells in spatial domains or annotations).
- n_{ij} is the number of elements that are in cluster i in the first partition and in cluster j in the second partition.
- a_i is the number of elements in cluster i of the first partition.
- b_j is the number of elements in cluster j of the second partition.
- $\binom{n}{2}$ is the binomial coefficient, representing the number of ways to choose 2 elements from n , calculated as:

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

Algorithm 3 Build distance matrix from KD-Tree

Inputs:

- Set of points P : a list of points in $\mathbf{R}^k$
- Integer k : number of nearest neighbors to consider for KNN (if used, $k > 0$)
- Float ϵ : maximum distance threshold (if used, $\epsilon > 0$)

Outputs:

- Sparse distance matrix D : a structure holding non-zero distances

```
1:  $T \leftarrow \text{KD-Tree}(P)$  ▷ Construct the KD-Tree from points
2: Initialize an empty sparse matrix  $D$ 
3: for each point  $p_i \in P$  do
4:   if  $k > 0$  then ▷ Only kNN specified
5:     neighbors  $\leftarrow \text{kNN\_search}(T, p_i, k)$ 
6:   else if  $\epsilon > 0$  then ▷ Only distance threshold specified
7:     neighbors  $\leftarrow \text{range\_search}(T, p_i, \epsilon)$ 
8:   else
9:     continue ▷ Skip this point if neither kNN nor threshold is specified
10:  for each neighbor  $p_j$  in neighbors do
11:     $d \leftarrow \text{Distance}(p_i, p_j)$  ▷ Calculate distance
12:    if  $k > 0$  or  $d < \epsilon$  then
13:      Add the distance  $d$  to the sparse matrix  $D$  at position  $(i, j)$ 
  return  $D$  ▷ Return the sparse distance matrix
```

Algorithm 4 query_ball_point

Inputs:

- *tree*: the constructed KD-Tree
- Point p : the center point for the search
- Float r : radius for searching neighbors

Outputs:

- List *indices*: indices of points within the radius r

```
1: Initialize an empty list indices
2: function QUERY_BALL_POINT(tree, p, r)
3:   neighbors  $\leftarrow \text{search\_within\_radius}(\textit{tree}, p, r)$  ▷ Search KD-Tree for neighbors
4:   for each neighbor  $n \in \textit{neighbors}$  do
5:     if  $\text{Distance}(p, n) < r$  then
6:       Append the index of  $n$  to indices
  return indices
```

Algorithm 5 local_density

Inputs:

- Set of coordinates *coords*: a list of points in $\mathbf{R}^k$
- Float *neighborhood_size*: radius for density calculation

Outputs:

- List *densities*: density values for each point in *coords*

```
1: tree  $\leftarrow$  KD-Tree(coords) ▷ Construct the KD-Tree from coordinates
2: Initialize an empty list densities
3: for each point p  $\in$  coords do
4:   indices  $\leftarrow$  query_ball_point(tree, p, neighborhood_size) ▷ Find neighbors within radius
5:   density  $\leftarrow \frac{\text{len}(\text{indices})}{\pi \cdot (\text{neighborhood\_size})^2}$  ▷ Calculate density
6:   densities += density
   return densities ▷ Return the list of densities
```

103 The ARI formula adjusts for the chance similarity between clusters by considering both pairwise agreements and
104 disagreements. The numerator counts the agreements, and the denominator normalizes the score, yielding an
105 index that ranges between -1 (no agreement) and 1 (perfect agreement), with 0 indicating random labeling.

106 1.3 CHAOS score calculation

107 The spatial continuity and compactness of each clustering result is quantified by the CHAOS[6] score in place of
108 the ARI. For the olfactory bulb dataset, the CHAOS score is used in place of ARI to assess cluster quality and is
109 detailed in Algorithm 6. The CHAOS score is based on the distances between points in a given cluster and their
110 nearest neighbors, capturing how spatially compact the clusters are. The CHAOS score provides a measure of how
111 spatially coherent the clusters are, with lower values indicating tighter and more compact clusters. The CHAOS
112 score can be described at a high-level as follows:

- 113 • **Standardize Locations:** the location data is standardized such that each spatial coordinate has zero mean
114 and unit variance.
- 115 • **1-Nearest Neighbor Calculation:** For each point *a* in cluster, we calculate the Euclidean distance to its
116 nearest neighbor within the cluster. This is done by constructing a 1-Nearest Neighbor (1NN) graph. See
117 Algorithm 7 for details.
- 118 • **Compute Cluster Distances:** For each cluster we sum the 1NN distances over all points in the cluster.
- 119 • **CHAOS Score Calculation:** The CHAOS score is then computed as the average 1NN distance across all
120 clusters, weighted by the number of points in each cluster.

121 1.4 Moran's I calculation

122 The spatial autocorrelation of each cluster is quantified using Moran's I[7], which assesses the degree of clustering of
123 similar values across a spatial distribution, with positive values indicating clustering of similar values and negative
124 values indicating dispersion. For the olfactory bulb dataset, Moran's I value is used in place of ARI to assess
125 cluster quality. The assignment of clusters is based on the proximity of spatial coordinates derived from k-nearest
126 neighbors (kNN). The Moran's I statistic calculation is detailed in Algorithm 8

127 1.5 Real world data preprocessing

128 **Mouse Ovary:** The dataset was provided directly by the authors [8] and was already preprocessed. RASP was
129 applied directly to the `AnnData` object provided. Briefly, processing included cell segmentation (`CellPose` and

Algorithm 6 CHAOS Function

Inputs:

- `clusterlabel`: Array of cluster labels for each data point.
- `L`: Array of spatial locations corresponding to each data point.

Outputs:

- **CHAOS**: The CHAOS score, representing the spatial compactness and continuity of the clusters.

```
1: NAs ← indices where isna(clusterlabel)    ▷ Identify indices of NA (null) values in cluster
   labels.
2: if length(NAs) > 0 then
3:   clusterlabel ← delete(clusterlabel, NAs)  ▷ Remove NA values from cluster label array.
4:   L ← delete(L, NAs)                      ▷ Remove corresponding locations for deleted cluster labels.
5: L ← scale(L)                             ▷ Standardize location data for better numerical stability.
6: unique_labels ← unique(clusterlabel)    ▷ Extract unique cluster labels from the cluster label
   array.
7: dist_val ← zeros(length(unique_labels))    ▷ Initialize an array to hold distance values for
   each unique cluster.
8: for each cluster  $k \in \text{unique\_labels}$  do
9:    $L_k \leftarrow L[\text{clusterlabel} == k]$     ▷ Extract locations corresponding to cluster  $k$ .
10:  if  $L_k$  has only one point then
11:    Continue                               ▷ Skip cluster if it consists of only one point.
12:  for each index  $i \in [0, \text{length}(L_k) - 1]$  do
13:     $\text{results}[i] \leftarrow \text{fx\_1NN}(i, L_k)$   ▷ Compute the nearest neighbor distance for point at index  $i$ .
14:     $\text{dist\_val}[\text{count}] \leftarrow \text{sum}(\text{results})$   ▷ Sum the 1NN distances for points in cluster  $k$ .
15:  $\text{dist\_val} \leftarrow \text{dist\_val}[\text{no NaN values}]$   ▷ Remove any NaN values from the distance array.
16: Return CHAOS ←  $\frac{\sum \text{dist\_val}}{\text{length}(\text{clusterlabel})}$     ▷ Compute and return the average CHAOS score.
```

Algorithm 7 1-Nearest Neighbor Function

Inputs:

- i : Index of the point for which to compute the nearest neighbor.
- L_k : Array of spatial locations corresponding to points in the cluster.

Outputs:

- $d_{i,1}$: Euclidean distance from point at index i to its nearest neighbor within the cluster.

```
1: distances ← pairwise_distance([ $L_k[i]$ ],  $L_k$ )  ▷ Compute distance between the point at index  $i$  and
   all points in  $L_k$ .
2: nearest_neighbor ← copy(distances)             ▷ Create a copy of the distances to sort them.
3: nearest_neighbor ← partition(nearest_neighbor, 1)  ▷ Partially sort distances such that the
   1st nearest neighbor distance is in the correct position.
   return  $d_{i,1} \leftarrow \text{nearest\_neighbor}[0, 1]$     ▷ Return the distance to the nearest neighbor.
```

Algorithm 8 Moran’s I Calculation

Inputs:

- C : Set of clusters in the dataset.
- L : Array of spatial coordinates corresponding to each data point in the dataset.
- Y : Vector of values assigned to each point based on their cluster.
- k : Number of nearest neighbors for kNN graph construction.

Outputs:

- I : Moran’s I statistic, representing the degree of spatial autocorrelation.

```
1:  $w \leftarrow \text{kNN}(L, k)$   ▷ Construct kNN graph where  $w$  represents the weight matrix of the neighbors.
2: for each point  $i$  in the dataset do
3:   for each neighbor  $j$  in  $w$  do
4:     if  $j$  is one of the  $k$  nearest neighbors of  $i$  then
5:        $w_{ij} \leftarrow 1$   ▷ Assign weight of 1 for neighbor  $j$ .
6:     else
7:        $w_{ij} \leftarrow 0$   ▷ Assign weight of 0 if not a neighbor.
8:  $\bar{Y} \leftarrow \frac{1}{N} \sum_{i=1}^N Y_i$   ▷ Calculate the mean of the values in vector  $Y$ .
9:  $I \leftarrow \frac{N}{\sum_{i=1}^N \sum_{j=1}^N w_{ij}} \cdot \frac{\sum_{i=1}^N \sum_{j=1}^N w_{ij} (Y_i - \bar{Y})(Y_j - \bar{Y})}{\sum_{i=1}^N (Y_i - \bar{Y})^2}$   ▷ Compute  $I$  using the provided formula.
   return  $I$   ▷ Return the calculated Moran’s I statistic.
```

130 **MERlin**[9, 10] for the acquisition of cell and transcript data. Cells with fewer than 10 transcript counts were
131 excluded from further analysis to ensure data quality. Data was log-normalized, scaled, PCA was performed,
132 neighborhood identification, and cell clusterings calculated. To determine the cell type for each cluster, transcript
133 counts were compiled for each gene across clusters, focusing on known markers within the top 10 identified
134 transcripts. Differential expression analysis was conducted using **Seurat** v3 in **R**, allowing the identification of
135 specific markers associated with each Leiden cluster. Additionally, visualization of spatial regions was achieved
136 using **Squidpy** [11] in conjunction with the **AnnData** [12] and **Scanpy** [13] libraries.

137 **DLPFC:** We downloaded the raw DLPFC data (slide #151673) from the spatialLIBD website ([http://](http://research.libd.org/spatialLIBD/index.html)
138 research.libd.org/spatialLIBD/index.html). The dataset was loaded into **R** using **Seurat**’s **CreateSeuratObject**
139 function with the `min.cells = 20` and `min.features = 20` parameters. Data were processed using the **SCTransform**
140 pipeline with the following parameters: `variable.features.n = NULL`, `variable.features.rv.th = 1.3`, `return.only.var.genes = FALSE`. Data were then converted to the h5ad file format using **SeuratDisk** package’s **SaveH5Seurat** and
141 **Convert** functions to be processed via RASP in **Python**.

142 **Mouse Olfactory bulb:** The dataset was downloaded from the SEDR publication GitHub repository [14]
143 (https://github.com/JinmiaoChenLab/SEDR_analyses/tree/master/data). Data were processed in **Python**
144 using **Scanpy** unless otherwise specified. The standard processing pipeline was run, including the following
145 functions: `calculate_qc_metrics`, `filter_genes` with `min_cells = 50`, `highly_variable_genes` with `flavor = 'seurat'`,
146 `normalize_total`, and `log1p`. The processed data were then analyzed using RASP.

147 **Breast Cancer:** We downloaded the Breast cancer Xenium dataset from the **SubcellularSpatialData R**
148 package via **ExperimentHub**, dataset # EH8567, sample ID 'IDC' ([https://www.bioconductor.org/packages/](https://www.bioconductor.org/packages/release/data/experiment/html/SubcellularSpatialData.html)
149 [release/data/experiment/html/SubcellularSpatialData.html](https://www.bioconductor.org/packages/release/data/experiment/html/SubcellularSpatialData.html)). Transcripts at each subcellular location were
150 assigned to cells using the `tx2spe` function with `bin = 'cell'`. The resulting **SingleCellExperiment** object was
151 converted to a *Seurat* object, and then converted to an h5ad file using **SeuratDisk** package’s **SaveH5Seurat** and
152 **Convert** functions. Downstream analysis was done in **Python** using the **Scanpy** and **Squidpy** packages. Mito-
153 chondrial, ribosomal, and hemoglobin genes identified with the `var_names.str.startswith("MT-"), var_names.str.starts`
154 commands, quality control metrics were calculated with the `calculate_qc_metrics` command, and data were
155 filtered using the `filter_cells` command where `min_genes = 20` and `min_cells = 3`. The filtered data were then
156 log normalized and the top 3000 most highly variable genes were identified using the `highly_variable_genes`

158 function with *favor = 'seurat'*. The processed data were then analyzed using RASP.

Table 1: Datasets used for RASP evaluation

Platform	Tissue	Organism	n cell/n location	n genes	Ground truth annotation?
MERFISH	Ovary	Mouse	43,038	228	True
Stereo-Seq	Brain (Olfactory bulb)	Mouse	19,109	14,367	False
10x Xenium	Breast cancer	Human	565,916	541	True
10x Visium	Brain (DLPFC)	Human	3,638	15,124	True
SRTsim	Simulation 1	NaN	10,000	150	True
SRTsim	Simulation 2	NaN	10,000	150	True

159 1.6 Simulated dataset processing

160 Raw simulated count data were processed in **Python** using the **Scanpy** and **Squidpy** packages. Count matrices
161 were normalized to medial total counts using the `pp.normalize_total` command, logarithmized with the `pp.log1p`
162 command and variable genes identified using the `pp.highly_variable_genes` function, all with default parameters.
163 The processed data were then analyzed using RASP. Note: for testing the **SpatialPCA** algorithm on the simulated
164 data, the raw simulated count matrix was processed via **Seurat's** `SCTransform` pipeline with default parameters,
165 as per the packages recommendation.

Table 2: Parameter Selection

Parameter Name	Algorithm	Type	Range	Default	Description	Recommendations
n.components	rSVD	int	5-100	20	Number of components to keep.	For spatial domain detection, use a range of 5 – 20 PCs to capture larger domains. For clustering tasks with higher resolution or heterogeneity, users may want to increase this parameter to a range of 30 – 60. Note that increased components will increase runtime.
n.oversamples	rSVD	int	2-10	10	Corresponds to the additional number of random vectors to sample the range of X to ensure proper conditioning.	Default is recommended for most situations. Smaller number can improve speed but negatively impact approximation quality. Users might want to increase this parameter up to $2k - n_{\text{components}}$ where k is the effective rank, for large matrices, noisy problems, matrices with slowly decaying spectrum, or to increase precision accuracy.

Continued on next page

Parameter Name	Algorithm	Type	Range	Default	Description	Recommendations
power_iteration_normalizer	rSVD	string	auto, QR, LU, none	auto	Power iteration normalizer for randomized SVD solver.	Default is recommended.
threshold	RASP	int	1-200	10	Corresponds to the kNN threshold used to calculate the sparse distance matrix used for spatial smoothing.	For larger spatial domains and tissue sections, a larger smoothing threshold between 50 and 100 is recommended. For high-resolution cell type tasks, kNNs between 3 and 10 are recommended. Users should incorporate a priori knowledge of tissue structure.
n_neighbors	RASP	int	5-50	10	Number of kNN when constructing a NN graph of the smoothed principal components, used by the clustering algorithms.	Default of 10 is recommended.
n_clusters	RASP	int	inf	10	The number of clusters identified by the clustering algorithm.	-
covariates	RASP	string, list	nan	None	Specification of additional covariates, should be in the adata.obs slot of the adata object.	Users can include additional features to incorporate into the reduction and clustering step such as the number of counts (library size), cellular density, chromatin accessibility, or protein abundance measures. Incorporation of these features should be tested against the RNA alone and results compared with CHAOS score and Moran's I.

Continued on next page

Parameter Name	Algorithm	Type	Range	Default	Description	Recommendations
covariate_kNN	RASP	int, list	1-200	None	If covariates are included, covariate smoothing is the kNN threshold to use for smoothing the additional feature(s), or threshold to calculate local density. This can be the same or different than the smoothing threshold feature.	If users are adding local cell density as a covariate, then a different smoothing threshold is recommended than that used for the RNA smoothing. For other covariates, keeping the same smoothing threshold as the RNA is recommended.
cluster_algorithm	Clustering	string	walktrap, louvain, mclust, louvain, leiden		Clustering algorithm utilized by RASP.	Clustering is largely dependent on the tissue type and structure users are hoping to identify. The walktrap algorithm works well for large, homogeneous structures, and evenly spaced spots such as 10x Visium, but has a large computational cost that increases runtime with larger datasets. Mclust utilizes GMM and generally performs consistently across all tissue types but is sensitive to the model type being fit. See the 'model.type' parameter. Louvain and Leiden algorithms outperform the other methods for high-resolution or heterogeneous labeling tasks such as identifying cell types or small interspersed tissue domains.
ground_truth_labels	Clustering	string	nan	None	Name of the ground truth labels in the adata.obs slot.	If the dataset is annotated, users can specify the data slot of the annotations and test the accuracy of RASP predictions. Used for benchmarking.

Continued on next page

Parameter Name	Algorithm	Type	Range	Default	Description	Recommendations
model_type	Mclust	string	See mclust-Model-Names for available model types	EEE	Corresponds to the GMM used by mclust.	Generally, EEE has the best performance on ST data.

2 Supplemental Results

2.1 A note on Moran’s I and CHAOS score

Many publications utilize Moran’s I and the CHAOS score as measures of cluster quality in the absence of ground truth annotations, where calculating the ARI is not feasible. This study employs these metrics for assessing the olfactory bulb dataset. To this end, we visualized both Moran’s I and CHAOS score over varying kNN thresholds from 1 to 50, and for β values ranging from 0 to 2 (see Extended Data Fig. 2, 4, 5, and 7).

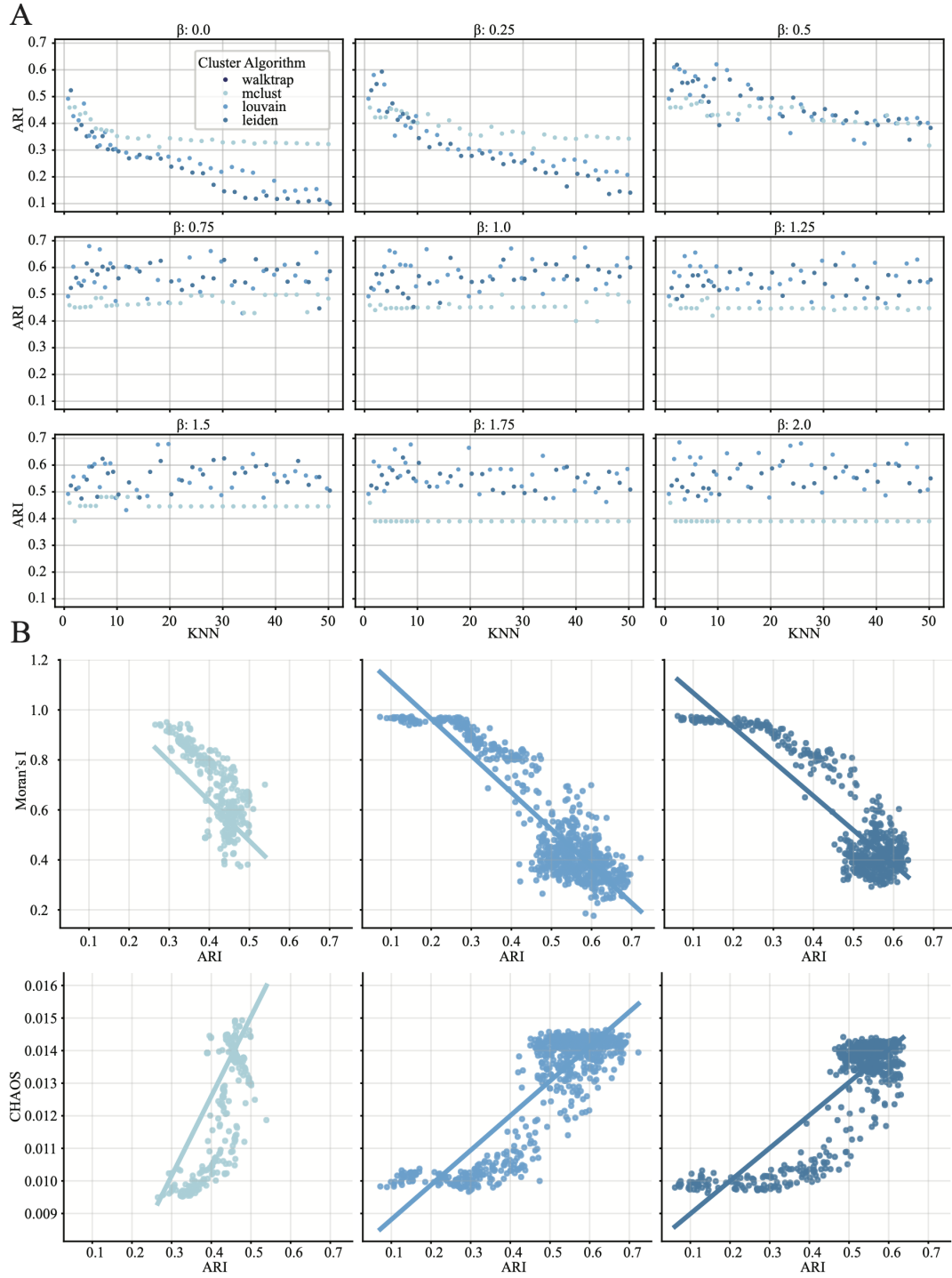
Across all datasets analyzed, the scores exhibited a similar trend: for small β values between 0 and 1, as the kNN threshold increased, the Moran’s I value rose logarithmically, while the CHAOS score decayed exponentially. In contrast, for larger β values from 1 to 2, the trends persisted but with lower Moran’s I values and higher CHAOS scores being observed. Notably, in the case of the ovary dataset, both the CHAOS and Moran’s I values remained invariant to changes in the kNN threshold when β exceeded 1, with scores in the other datasets trending towards little to no change as well.

Additionally, we visualized the relationship between ARI and Moran’s I, as well as between ARI and CHAOS score for all datasets with available ground truth annotations. Interestingly, for the DLPFC and ovary datasets, we found that Moran’s I and ARI were negatively correlated (see Extended Data Fig. 1,3). In contrast, the CHAOS score was positively correlated with ARI in the ovary dataset, regardless of the clustering algorithm used. However, in the DLPFC dataset, only the Louvain and Leiden clustering methods demonstrated a positive association with the CHAOS score, while Walktrap and MCLust did not exhibit such a relationship.

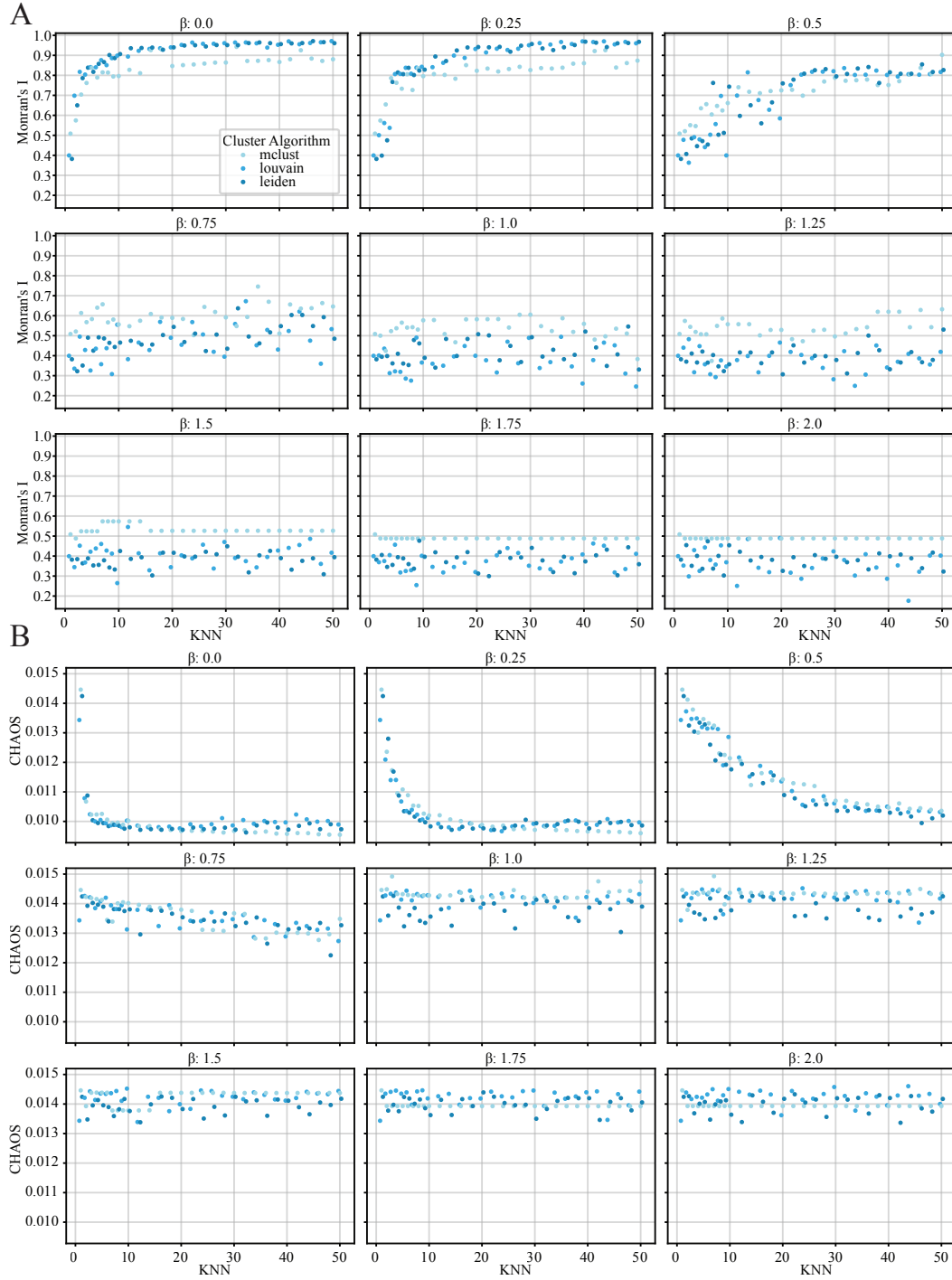
These results underscore both the importance and the limited utility of CHAOS score and Moran’s I in assessing cluster quality. Moreover, they suggest that the specific clustering task—such as distinguishing between cell types versus spatial domain annotations—and the choice of clustering algorithm will significantly impact the usefulness of these two metrics in evaluating cluster quality.

References

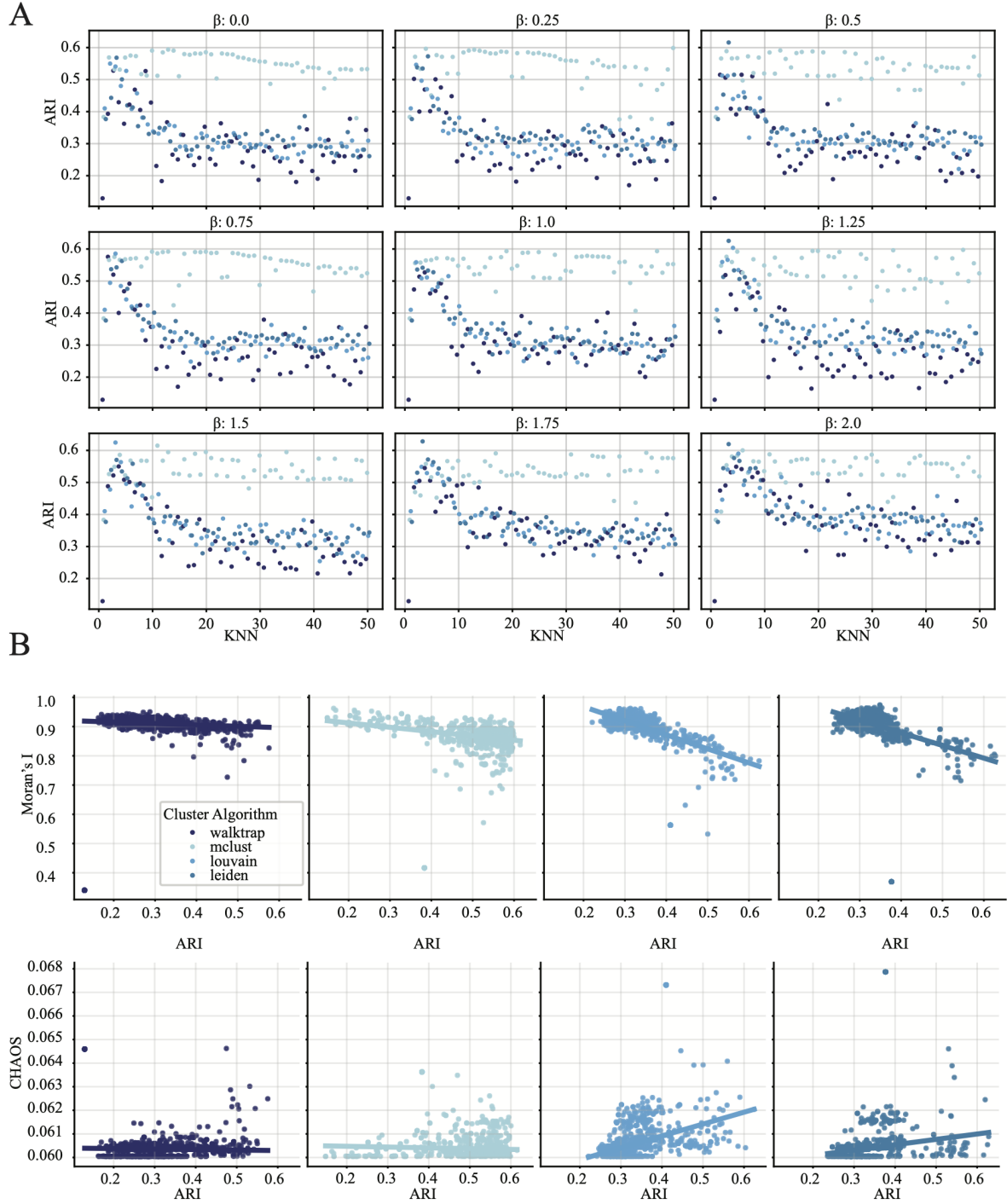
1. Erichson, N. B., Voronin, S., Brunton, S. L. & Kutz, J. N. Randomized Matrix Decompositions using R. *Journal of Statistical Software* **89**. ISSN: 1548-7660. arXiv: 1608.02148[stat]. <http://arxiv.org/abs/1608.02148> (2024) (2019).
2. Pedregosa, F. *et al.* Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* **12**, 2825–2830. <http://jmlr.org/papers/v12/pedregosa11a.html> (2011).
3. Martinsson, P.-G. & Tropp, J. A. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numerica* **29**, 403–572. ISSN: 0962-4929, 1474-0508. <https://www.cambridge.org/core/journals/acta-numerica/article/randomized-numerical-linear-algebra-foundations-and-algorithms/4486926746CFF4547F42A2996C7DC09C> (2024) (May 2020).



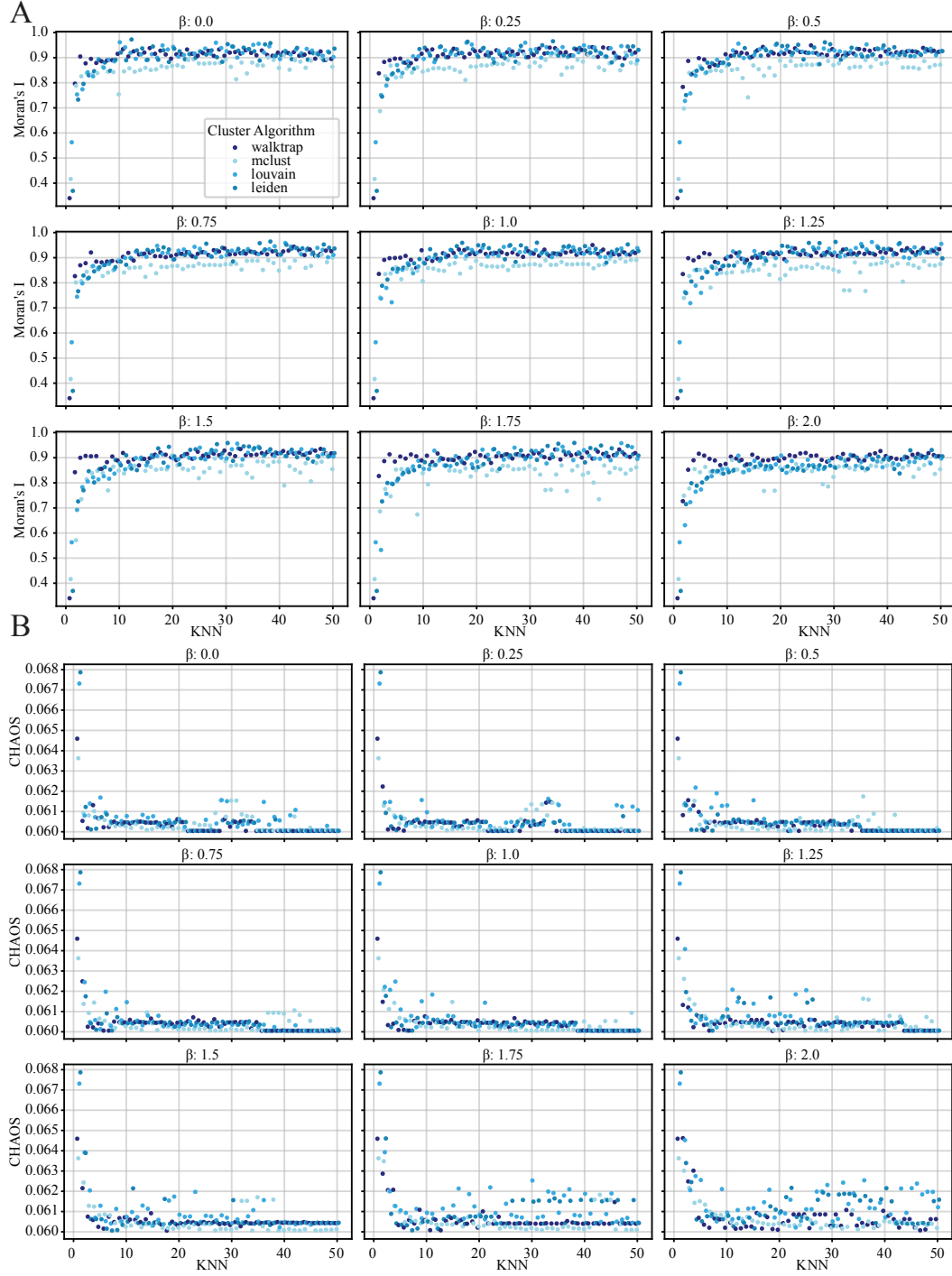
Extended Data Fig. 1: **Mouse Ovary supplement 1. A:** ARI values plotted against kNN distance threshold. Colors indicate clustering algorithm, each subplot corresponds to a distinct β value. **B:** Moran's I and CHAOS values plotted against ARI value, colors indicate clustering algorithm. Lines represent robust linear regression best fit.



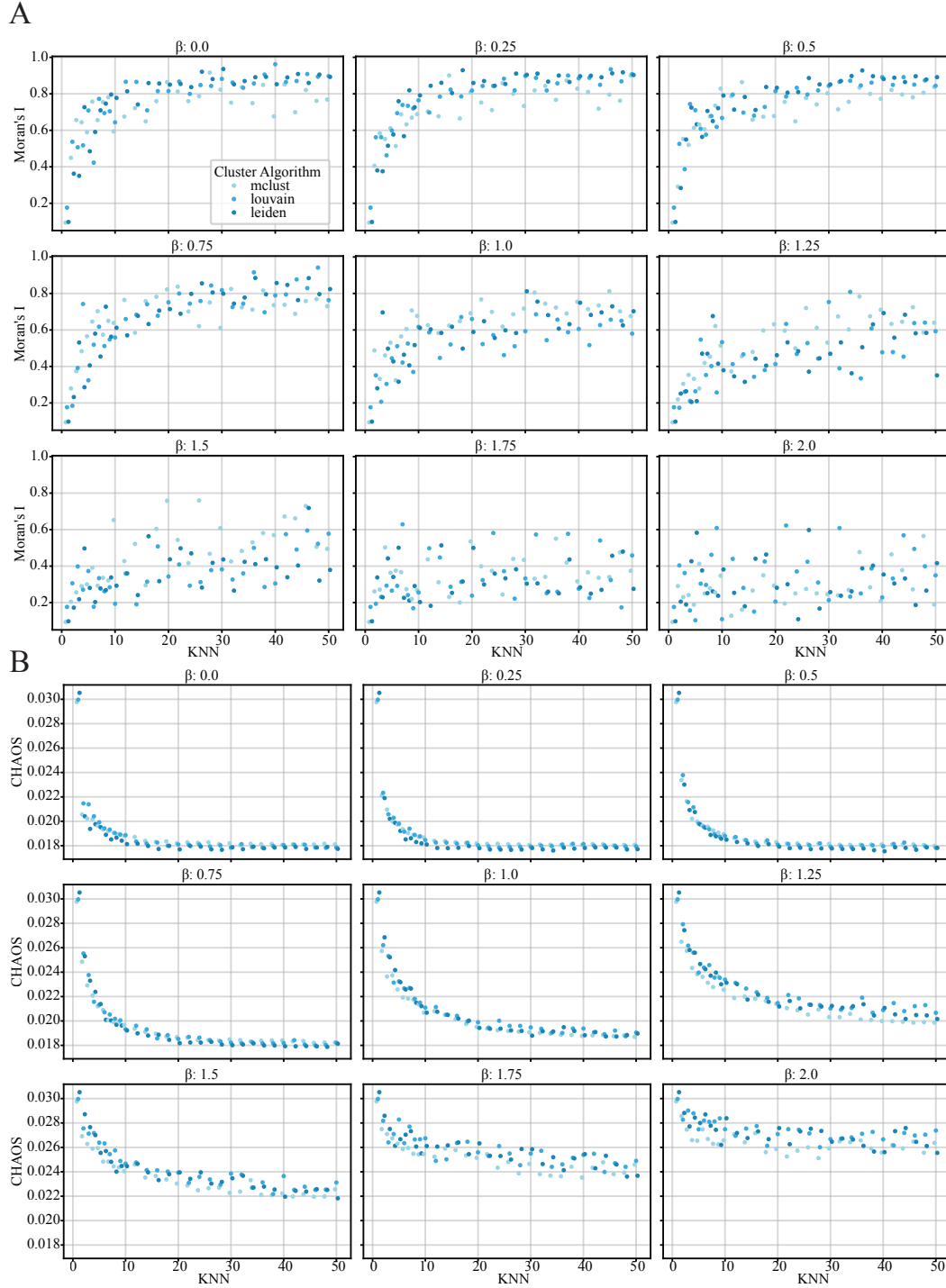
Extended Data Fig. 2: **Mouse Ovary supplement 2.** **A:** Moran's I statistic plotted against kNN distance threshold. Colors indicate clustering algorithm, each subplot corresponds to a distinct β value. **B:** CHAOS score plotted against kNN distance threshold. Colors indicate clustering algorithm, each subplot corresponds to a distinct β value.



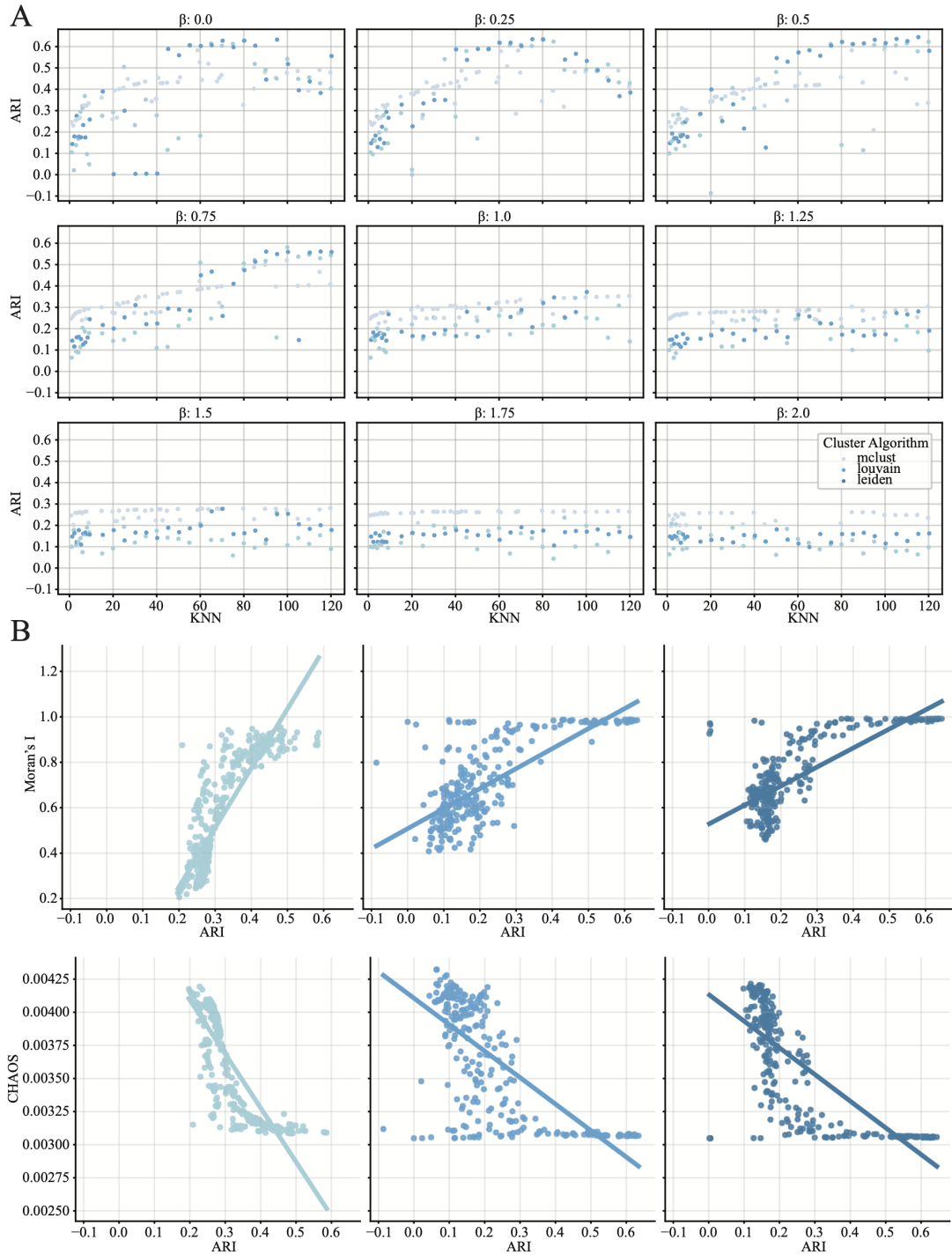
Extended Data Fig. 3: **DLPFC supplement 1. A:** ARI values plotted against kNN distance threshold. Colors indicate clustering algorithm, each subplot corresponds to a distinct β value. **B:** Moran's I and CHAOS values plotted against ARI value, colors indicate clustering algorithm. Lines represent robust linear regression best fit.



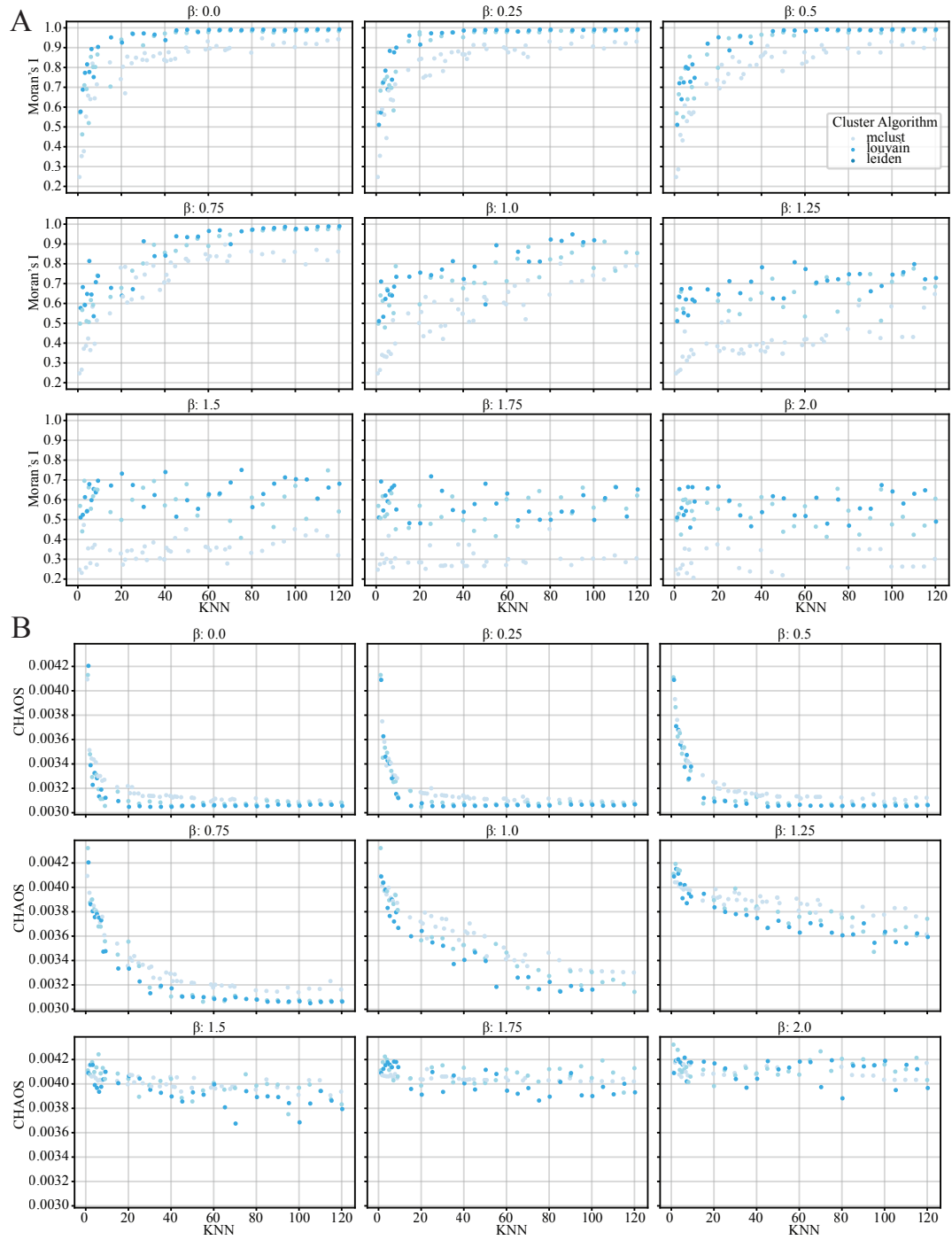
Extended Data Fig. 4: **DLPFC supplement 2. A:** Moran's I statistic plotted against kNN distance threshold. Colors indicate clustering algorithm, each subplot corresponds to a distinct β value. **B:** CHAOS score plotted against kNN distance threshold. Colors indicate clustering algorithm, each subplot corresponds to a distinct β value.



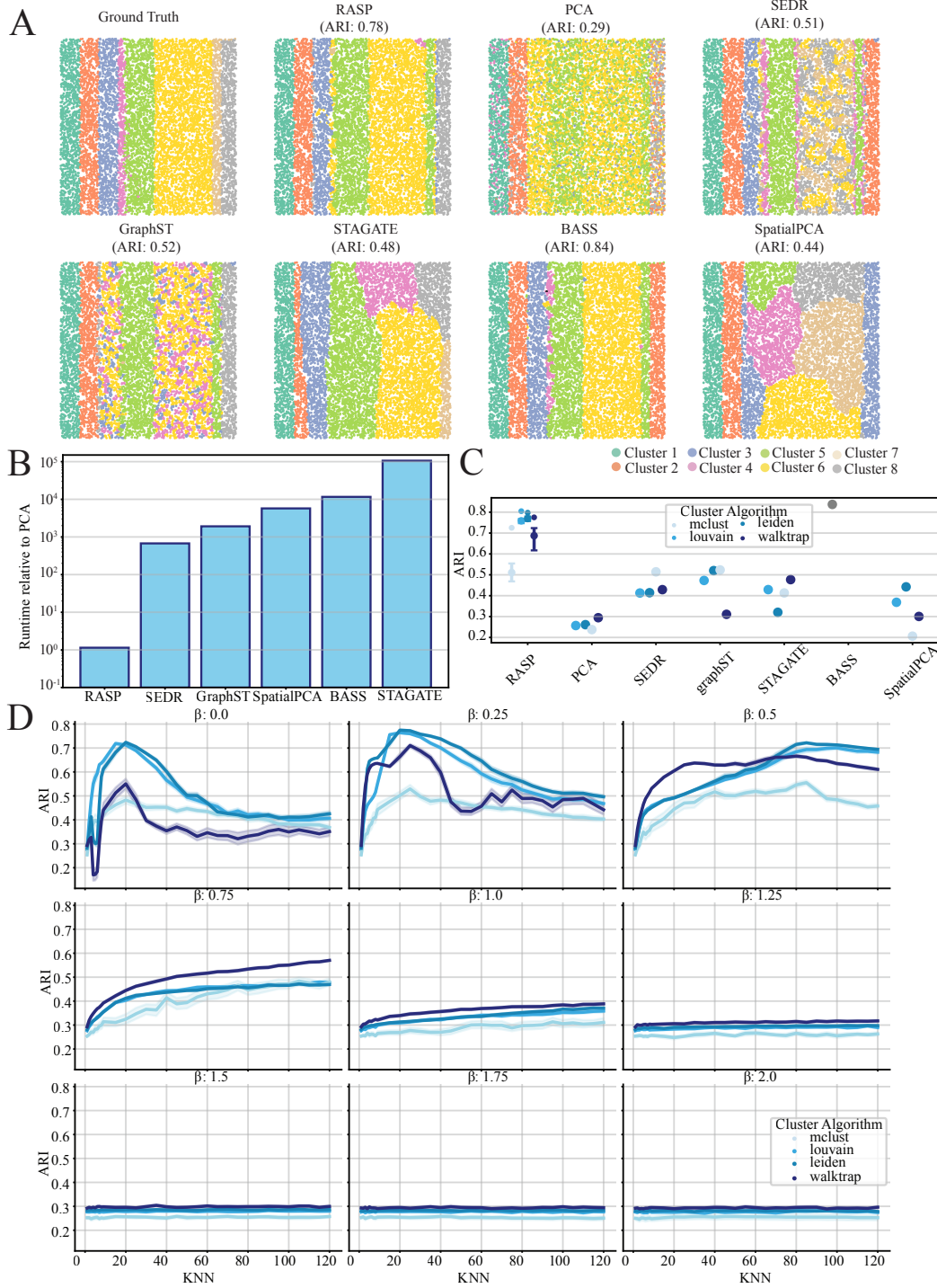
Extended Data Fig. 5: **Olfactory bulb supplement.** **A:** Moran's I statistic plotted against kNN distance threshold. Colors indicate clustering algorithm, each subplot corresponds to a distinct β value. **B:** CHAOS score plotted against kNN distance threshold. Colors indicate clustering algorithm, each subplot corresponds to a distinct β value.



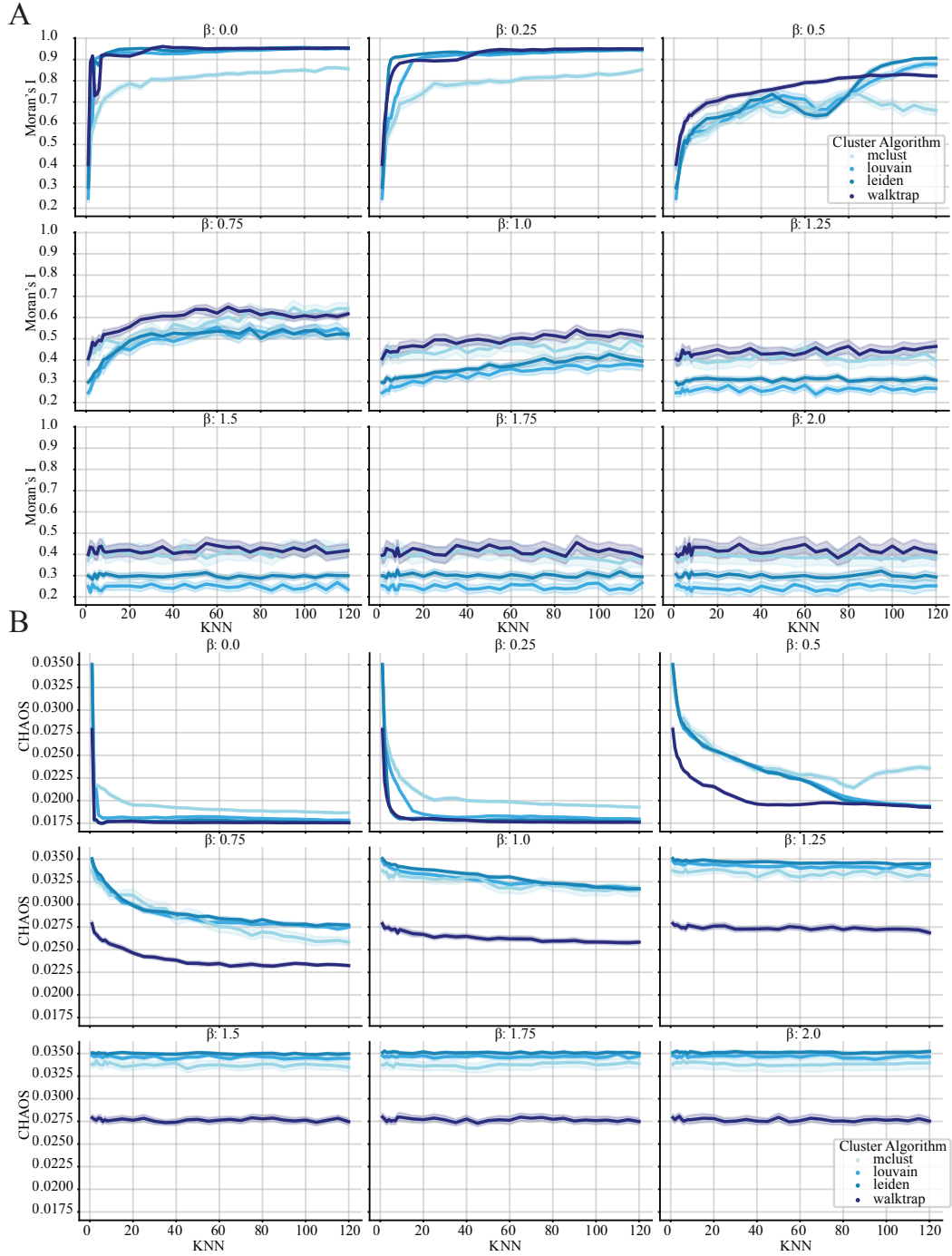
Extended Data Fig. 6: **Xenium supplement 1. A:** ARI values plotted against kNN distance threshold. Colors indicate clustering algorithm, each subplot corresponds to a distinct β value. **B:** Moran's I and CHAOS values plotted against ARI value, colors indicate clustering algorithm.



Extended Data Fig. 7: **Xenium supplement 2. A:** Moran's I statistic plotted against kNN distance threshold. Colors indicate clustering algorithm, each subplot corresponds to a distinct β value. **B:** CHAOS score plotted against kNN distance threshold. Colors indicate clustering algorithm, each subplot corresponds to a distinct β value. Lines represent robust linear regression best fit.

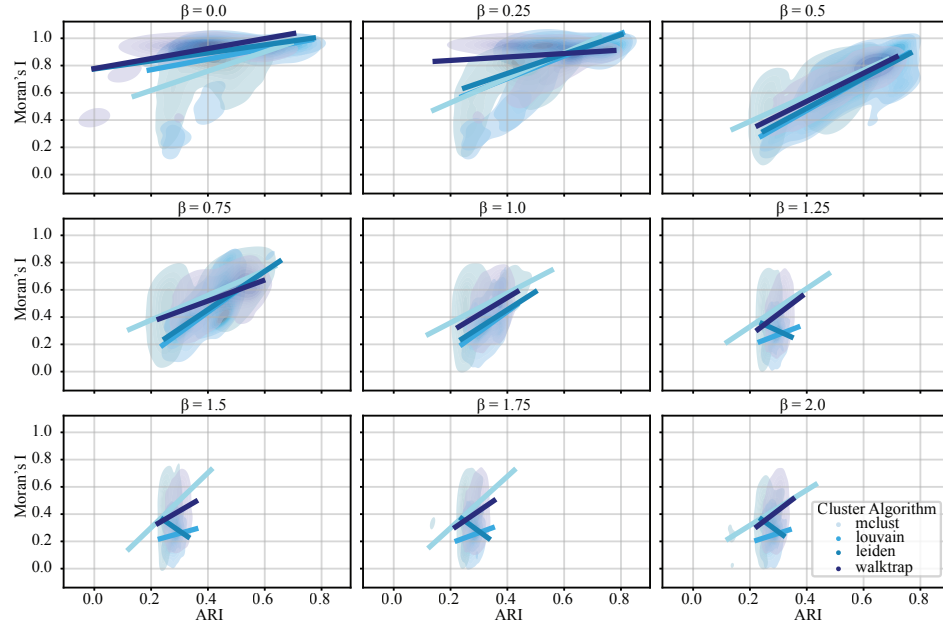


Extended Data Fig. 8: **Simulation 1 (Stripes)** **A:** Ground truth annotations (left) and corresponding spatial domains identified by RASP, PCA, and other methods. **B:** Quantification of runtime for all methods compared to normal PCA. **C:** Quantification of ARI for all methods, different colors indicate the clustering algorithm used to assign labels. Interquartile range and median ARI values at default RASP parameters ($kNN = 15-30$, $\beta = 0.25$) is shown, along with maximum ARI values achieved by RASP. **D:** ARI values for RASP evaluated on 100 *Stripe* replicates plotted against kNN , each subplot represents a distinct β value. Colored line represents the median, shaded colors indicate the interquartile range. Colors correspond to clustering algorithm.

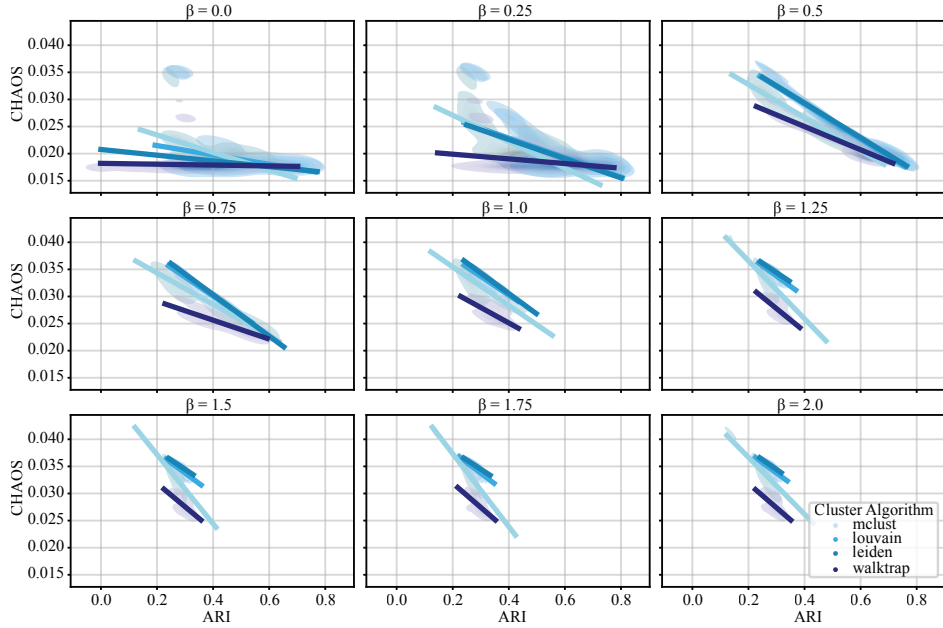


Extended Data Fig. 9: **Simulation 1 (Stripes)** **A:** Moran's I statistic plotted against kNN distance threshold. Colors indicate clustering algorithm, each subplot corresponds to a distinct β value. **B:** CHAOS score plotted against kNN distance threshold. Colored line represents the median, shaded colors indicate the interquartile range. Colors correspond to clustering algorithm, each subplot corresponds to a distinct β value.

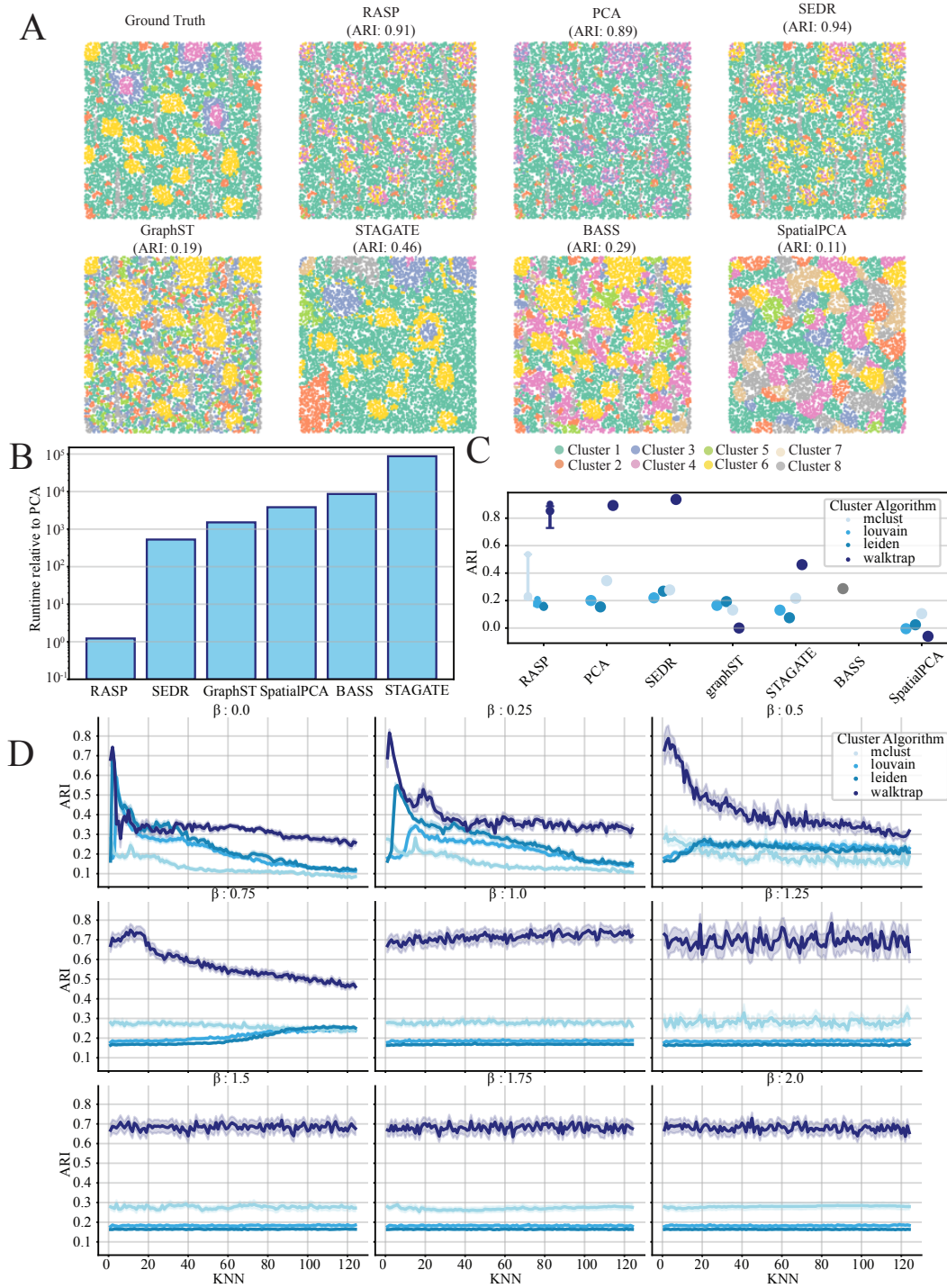
A



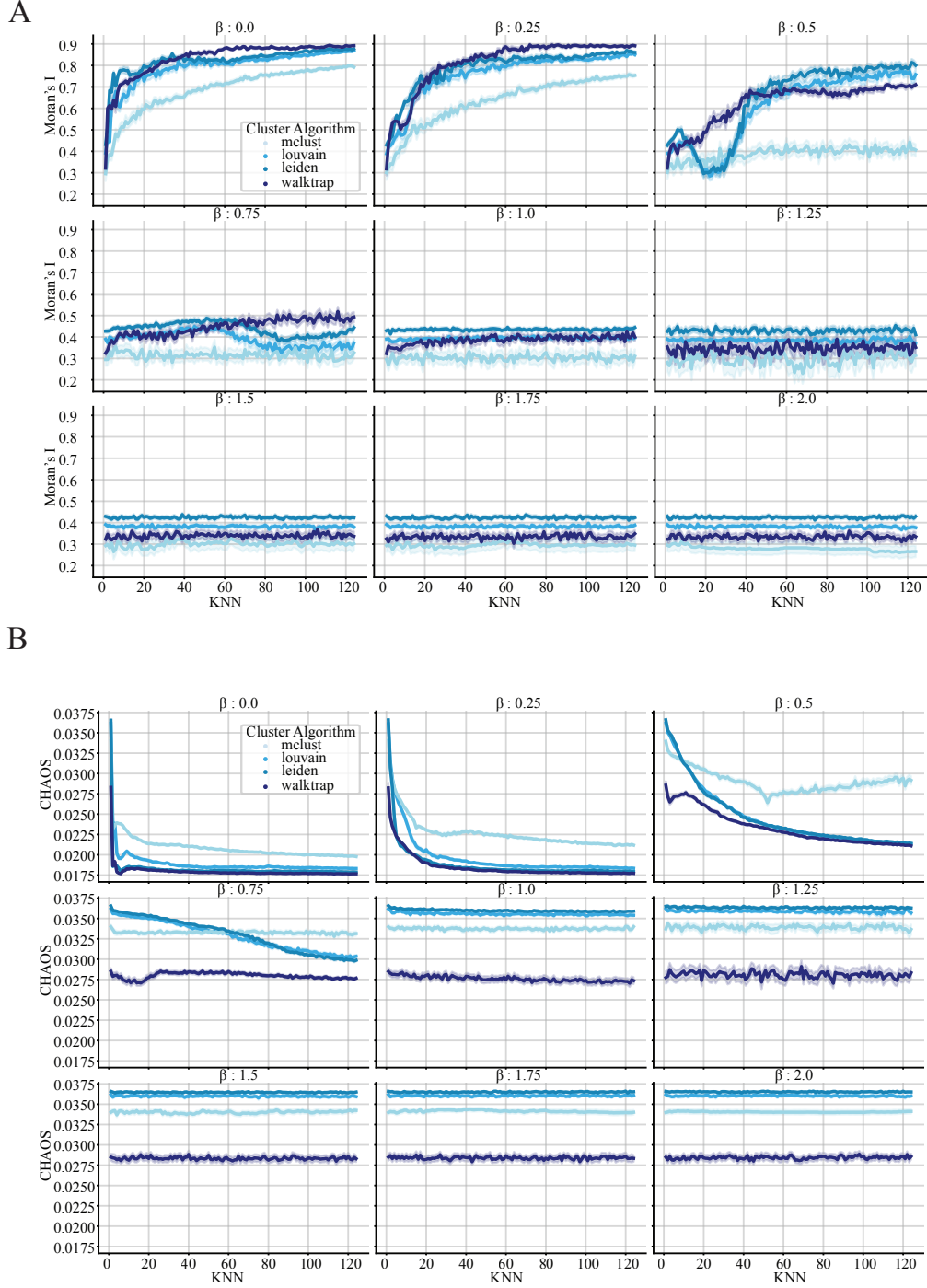
B



Extended Data Fig. 10: **Simulation 1 (Stripes)** **A:** Bivariate kernel density estimate of Moran's I statistic against ARI. Colors indicate clustering algorithm, each subplot corresponds to a distinct β value. **B:** Bivariate kernel density estimate of CHAOS score plotted against ARI. Colors indicate clustering algorithm, each subplot corresponds to a distinct β value. Lines represent robust linear regression best fit.

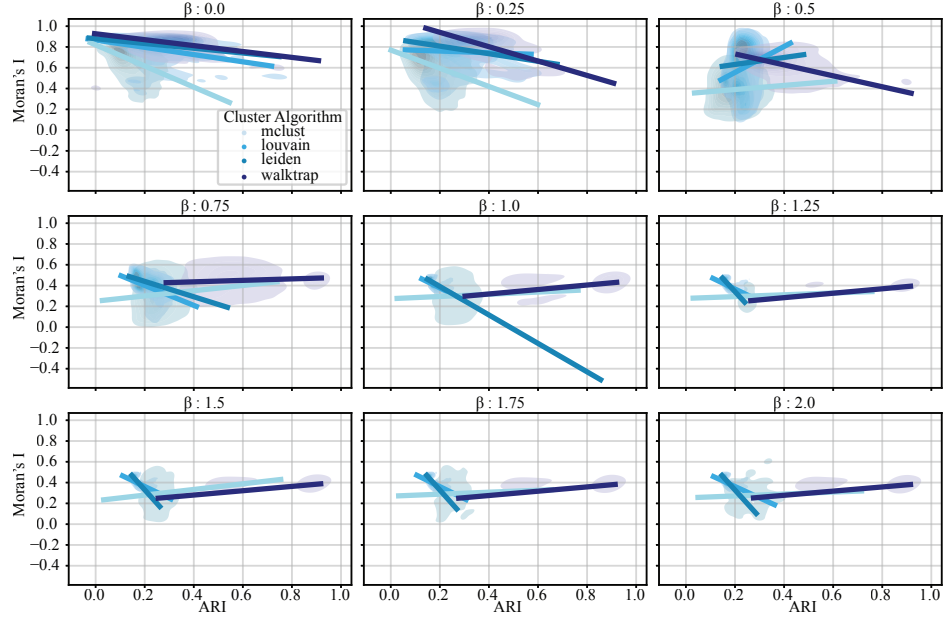


Extended Data Fig. 11: **Simulation 2 (Dots)** **A:** Ground truth annotations (left) and corresponding spatial domains identified by RASP, PCA, and other methods. **B:** Quantification of runtime for all methods compared to normal PCA. **C:** Quantification of ARI for all methods, different colors indicate the clustering algorithm used to assign labels. Interquartile range and median ARI values at default RASP parameters ($kNN = 3-20$, $\beta = 2$) is shown, along with maximum ARI values achieved by RASP. **D:** ARI values for RASP evaluated on 100 *Dot* replicates plotted against kNN , each subplot represents a distinct β value. Colors indicate clustering algorithm.

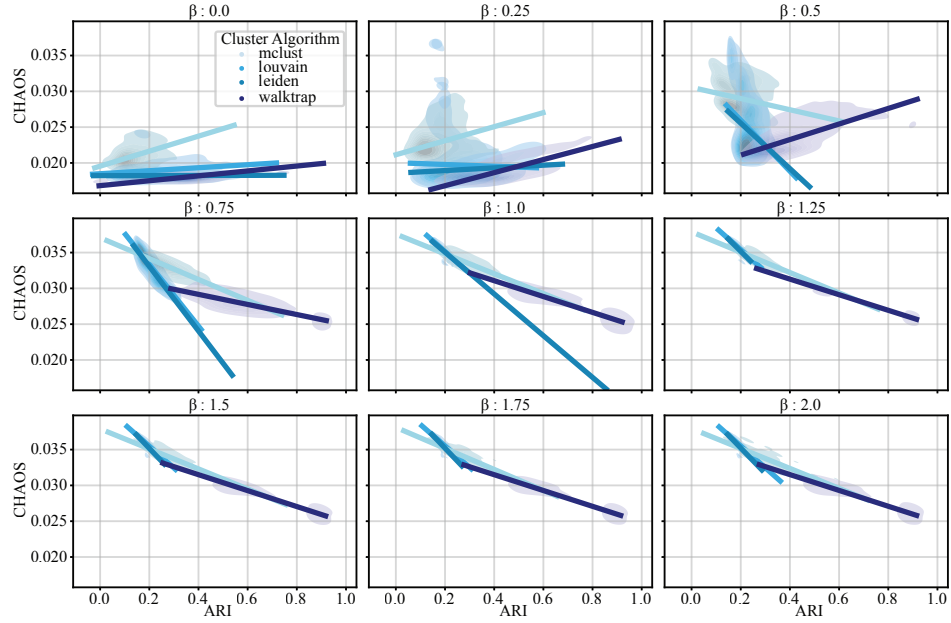


Extended Data Fig. 12: **Simulation 2 (Dots)** **A:** Moran's I statistic plotted against kNN distance threshold. Colors indicate clustering algorithm, each subplot corresponds to a distinct β value. **B:** CHAOS score plotted against kNN distance threshold. Colored line represents the median, shaded colors indicate the interquartile range. Colors correspond to clustering algorithm, each subplot corresponds to a distinct β value.

A

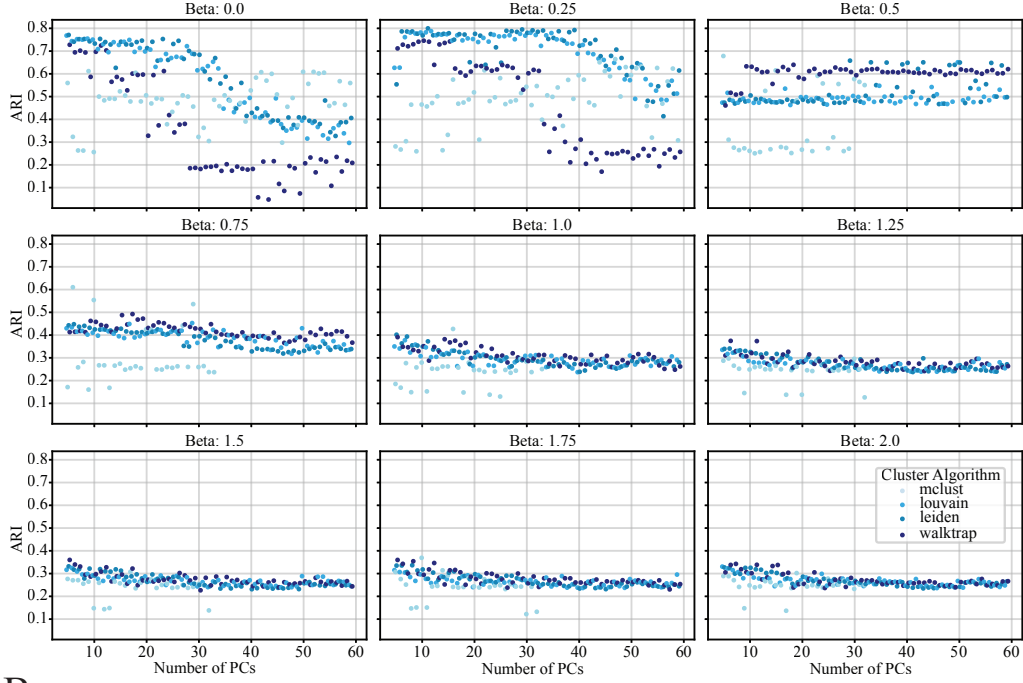


B

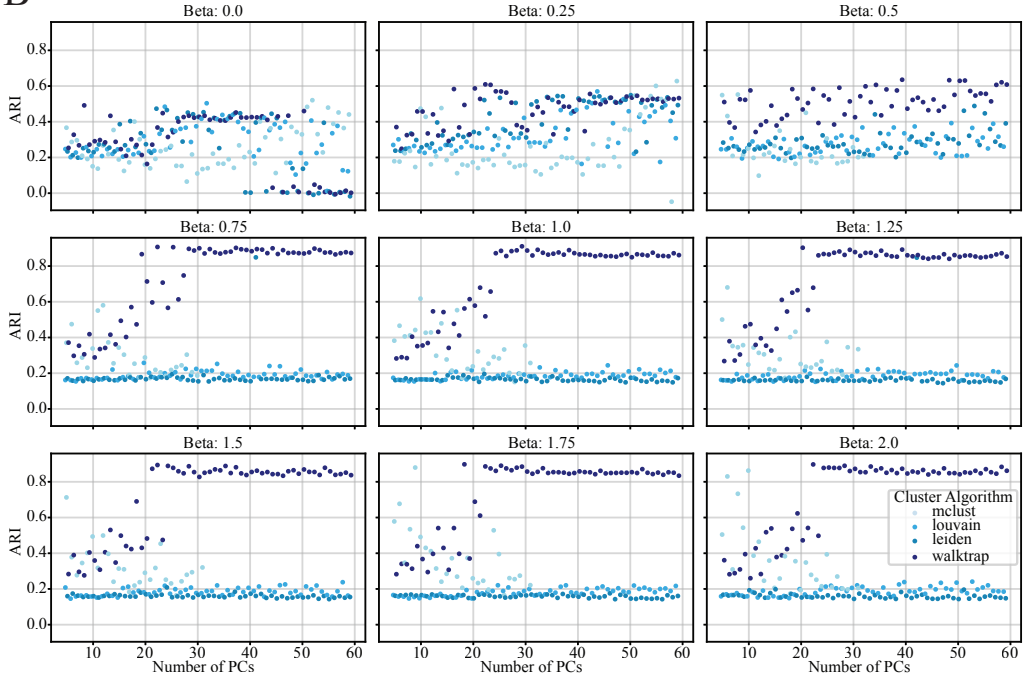


Extended Data Fig. 13: **Simulation 2 (Dots)** **A:** Bivariate kernel density estimate of Moran's I statistic against ARI. Colors indicate clustering algorithm, each subplot corresponds to a distinct β value. **B:** Bivariate kernel density estimate of CHAOS score plotted against ARI. Colors indicate clustering algorithm, each subplot corresponds to a distinct β value. Lines represent robust linear regression best fit.

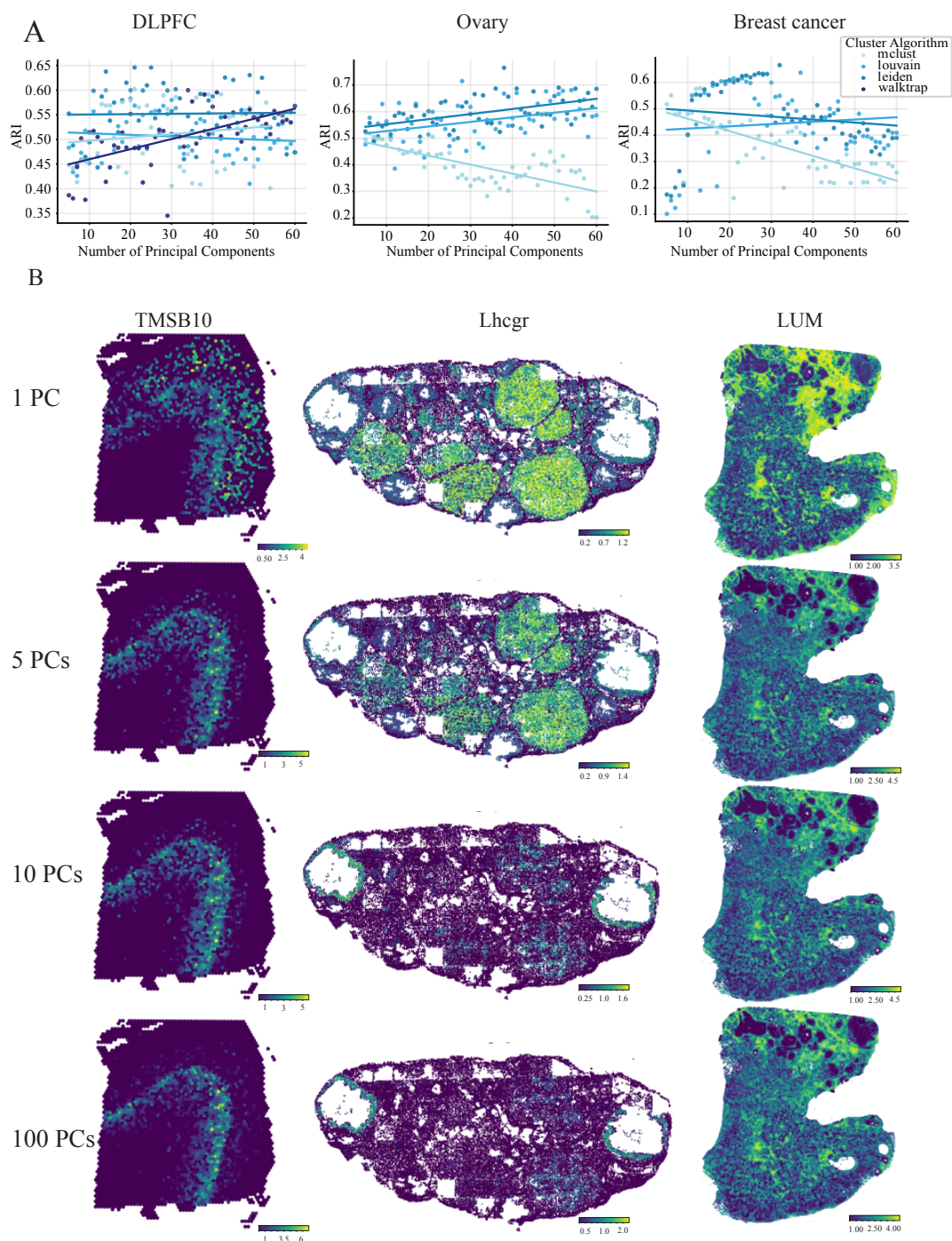
A



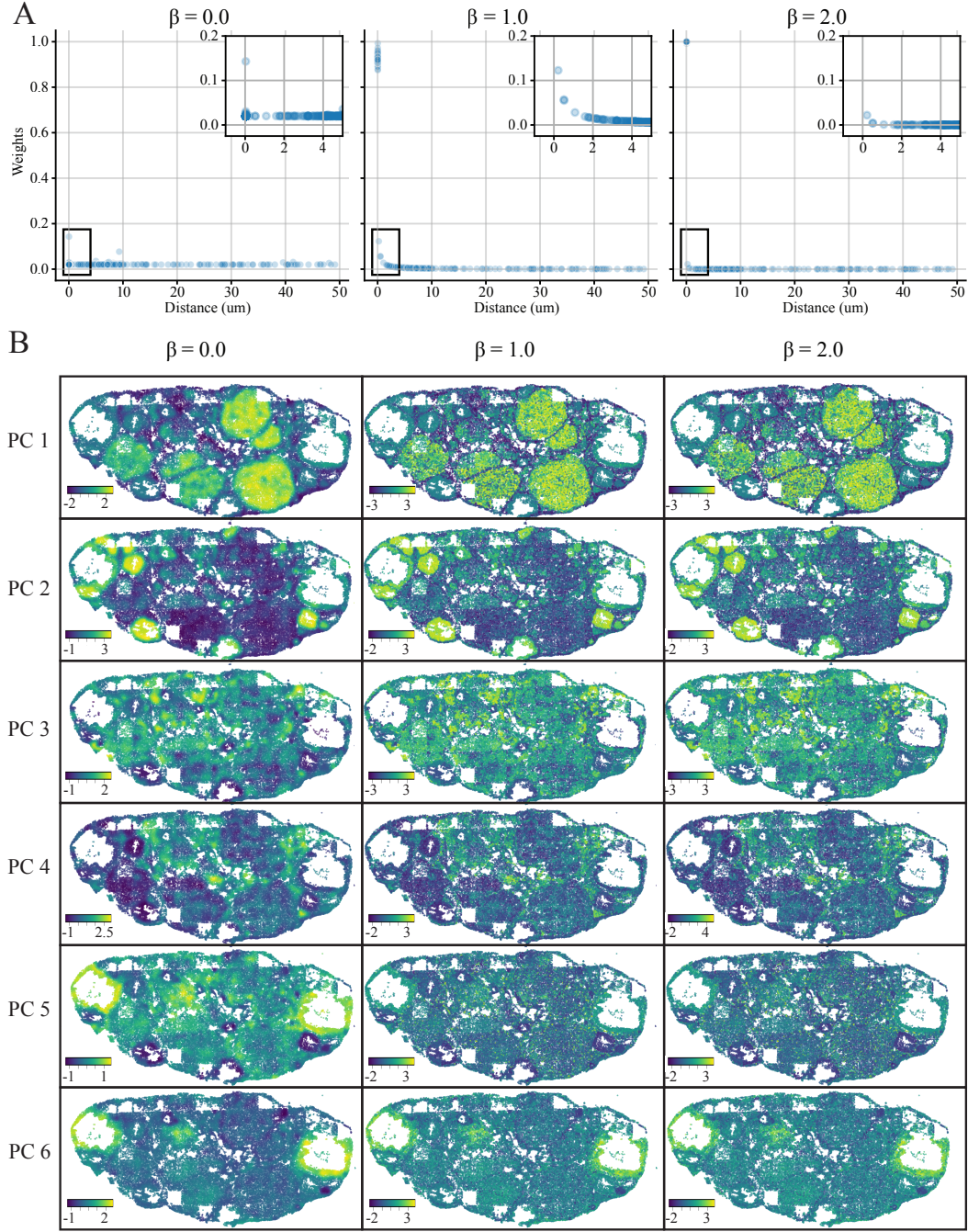
B



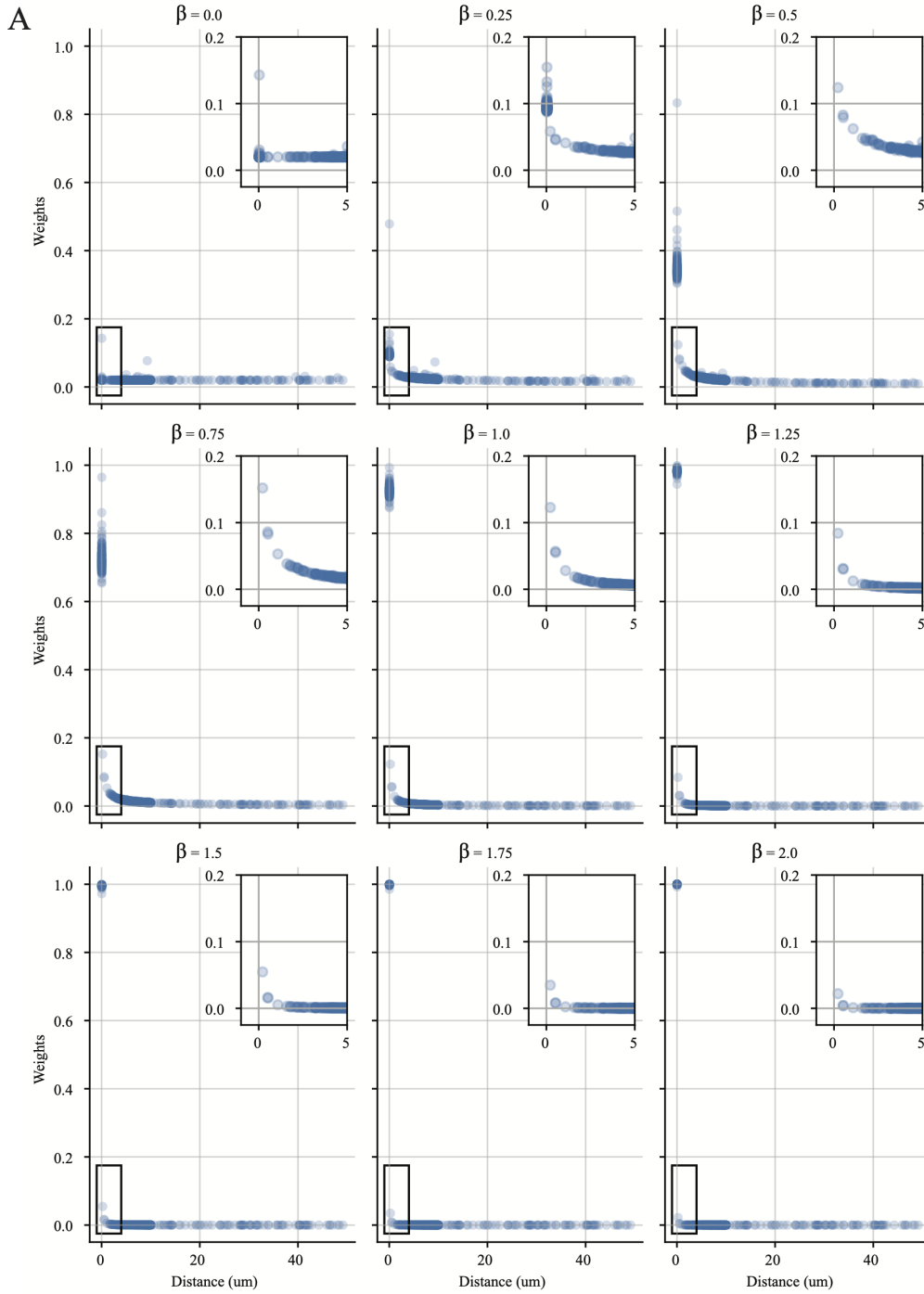
Extended Data Fig. 14: **PC sensitivity (simulated data)** **A:** ARI values plotted against number of PCs utilized by RASP for the *Stripes* dataset. Colors indicate clustering algorithm, each subplot corresponds to a distinct β value. **B:** ARI values plotted against number of PCs utilized by RASP for the *Dots* dataset. Colors indicate clustering algorithm, each subplot corresponds to a distinct β value.



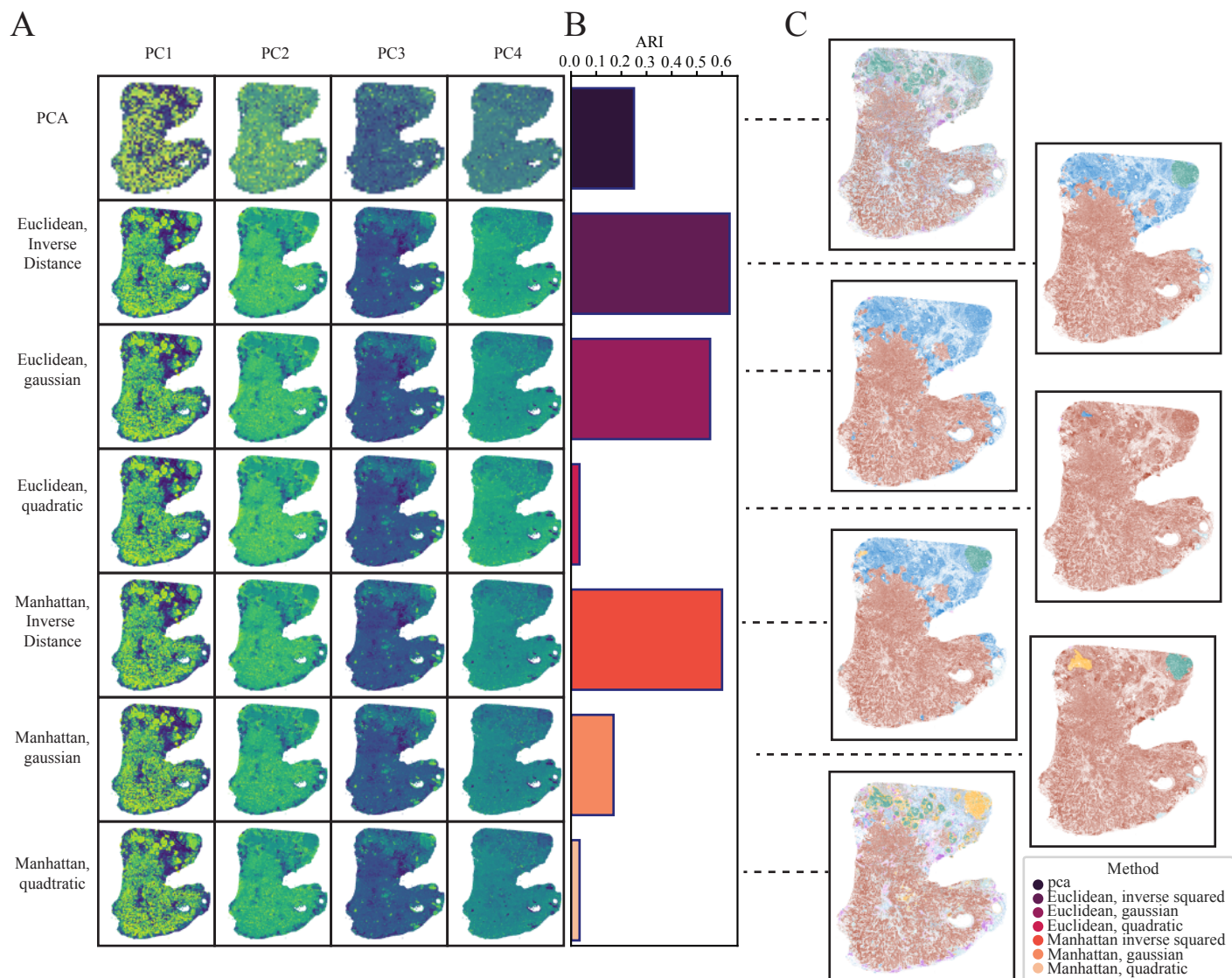
Extended Data Fig. 15: **PC Sensitivity Analysis.** **A:** ARI values plotted against number of PCs utilized by RASP for the DLPFC (left), Mouse ovary (middle) and Human breast cancer (right) datasets. Colors indicate clustering algorithm. **B:** Reduced rank reconstructed gene signatures for TMSB10 (left), Lhgr (middle), and LUM (right) at increasing ranks.



Extended Data Fig. 16: **Effects of inverse weighting on PCs at various β values.** **A:** Inverse weight values plotted against real world distances (um) for $\beta = 0$, $\beta = 1$, and $\beta = 2$. **B:** First six PCs for the mouse ovary dataset visualized spatially on the tissue, smoothed by inverse distances exponentiated by 0 (left), 1 (middle) and 2(right).



Extended Data Fig. 17: **Inverse weights at different β values.** **A:** Inverse weight values plotted against real world distances (um) at various values of β



Extended Data Fig. 18: **Differential distance measurements and distance weights applied to Human breast cancer dataset.** **A:** Normal and spatially smoothed PCs 1-4 visualized spatially on the tissue. **B:** RASP-identified domains from the different distance and weighting schemes presented in **A**. **C:** Kernel density estimates of the first 4 normal and spatially smoothed PCs. Colors indicate distance and weighting schemes, as shown in **A**. **D:** Quantification of domain detection by ARI score for the regions identified in **B**.

- 198 4. Virtanen, P. *et al.* SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*
199 **17**. Publisher: Nature Publishing Group, 261–272. ISSN: 1548-7105. <https://www.nature.com/articles/s41592-019-0686-2> (2024) (Mar. 2020).
- 201 5. Hubert, L. & Arabie, P. Comparing partitions. *Journal of Classification* **2**, 193–218. ISSN: 1432-1343. <https://doi.org/10.1007/BF01908075> (2024) (Dec. 1, 1985).
- 202 6. Alexandrov, T. & Bartels, A. Testing for presence of known and unknown molecules in imaging mass spectrom-
203 etry. *Bioinformatics* **29**, 2335–2342. ISSN: 1367-4803. <https://doi.org/10.1093/bioinformatics/btt388>
204 (2024) (Sept. 15, 2013).
- 205 7. Moran, P. A. P. Notes on Continuous Stochastic Phenomena. *Biometrika* **37**. Publisher: [Oxford University
206 Press, Biometrika Trust], 17–23. ISSN: 0006-3444. <https://www.jstor.org/stable/2332142> (2024) (1950).
- 207 8. Huang, R. *et al.* *Single-cell and spatiotemporal profile of ovulation in the mouse ovary* Pages: 2024.05.20.594719
208 Section: New Results. June 5, 2024. <https://www.biorxiv.org/content/10.1101/2024.05.20.594719v4>
209 (2024).
- 210 9. Stringer, C., Wang, T., Michaelos, M. & Pachitariu, M. Cellpose: a generalist algorithm for cellular seg-
211 mentation. *Nature Methods* **18**. Publisher: Nature Publishing Group, 100–106. ISSN: 1548-7105. <https://www.nature.com/articles/s41592-020-01018-x> (2024) (Jan. 2021).
- 212 10. Dalsasso, E., Rambour, C., Trouvé, N. & Thome, N. MERLIN-Seg: Self-supervised despeckling for label-
213 efficient semantic segmentation. *Computer Vision and Image Understanding* **241**, 103940. ISSN: 1077-3142.
214 <https://www.sciencedirect.com/science/article/pii/S1077314224000213> (2024) (Apr. 1, 2024).
- 215 11. Palla, G., Fischer, D. S., Regev, A. & Theis, F. J. Spatial components of molecular tissue biology. *Nature*
216 *Biotechnology* **40**. Number: 3 Publisher: Nature Publishing Group, 308–318. ISSN: 1546-1696. <https://www.nature.com/articles/s41587-021-01182-1> (2023) (Mar. 2022).
- 217 12. Virshup, I., Rybakov, S., Theis, F. J., Angerer, P. & Wolf, F. A. anndata: Access and store annotated data
218 matrices. *Journal of Open Source Software* **9**, 4371. ISSN: 2475-9066. <https://joss.theoj.org/papers/10.21105/joss.04371> (2024) (Sept. 16, 2024).
- 219 13. Wolf, F. A., Angerer, P. & Theis, F. J. SCANPY: large-scale single-cell gene expression data analysis. *Genome*
220 *Biology* **19**, 15. ISSN: 1474-760X. <https://doi.org/10.1186/s13059-017-1382-0> (2024) (Feb. 6, 2018).
- 221 14. Chen, S. *et al.* Integration of spatial and single-cell data across modalities with weakly linked features.
222 *Nature Biotechnology*. Publisher: Nature Publishing Group, 1–11. ISSN: 1546-1696. <https://www.nature.com/articles/s41587-023-01935-0> (2024) (Sept. 7, 2023).
- 223
- 224
- 225
- 226
- 227