quollr: An R Package for Visualizing 2-D Models from Nonlinear Dimension Reductions in High-Dimensional Space

by Jayani P. Gamage, Dianne Cook, Paul Harrison, Michael Lydeamore, and Thiyanga S. Talagala

Abstract Nonlinear dimension reduction methods provide a low-dimensional representation of high-dimensional data by applying a Nonlinear transformation. However, the complexity of the transformations and data structures can create wildly different representations depending on the method and hyper-parameter choices. It is difficult to determine whether any of these representations are accurate, which one is the best, or whether they have missed important structures. The R package quollr has been developed as a new visual tool to determine which method and which hyper-parameter choices provide the most accurate representation of high-dimensional data. The scurve data from the package is used to illustrate the algorithm. Single-cell RNA sequencing (scRNA-seq) data from mouse limb muscles are used to demonstrate the usability of the package.

1 Introduction

Nonlinear dimension reduction (NLDR) techniques, such as t-distributed stochastic neighbor embedding (tSNE) (Maaten and Hinton, 2008), uniform manifold approximation and projection (UMAP) (McInnes et al., 2018), potential of heat-diffusion for affinity-based trajectory embedding (PHATE) algorithm (Moon et al., 2019), large-scale dimensionality reduction Using triplets (TriMAP) (Amid and Warmuth, 2019), and pairwise controlled manifold approximation (PaCMAP) (Wang et al., 2021), can create hugely different representations depending on the selected method and hyper-parameter choices. It is difficult to determine whether any of these representations are accurate, which one is the best, or whether they have missed important structures.

This paper presents the R package, `quollr`, which is useful for understanding how NLDR warps high-dimensional space and fits the data. Starting with an NLDR layout, our approach is to create a 2-*D* wireframe model representation, that can be lifted and displayed in the high-dimensional (*p*-*D*) space (Figure 1).

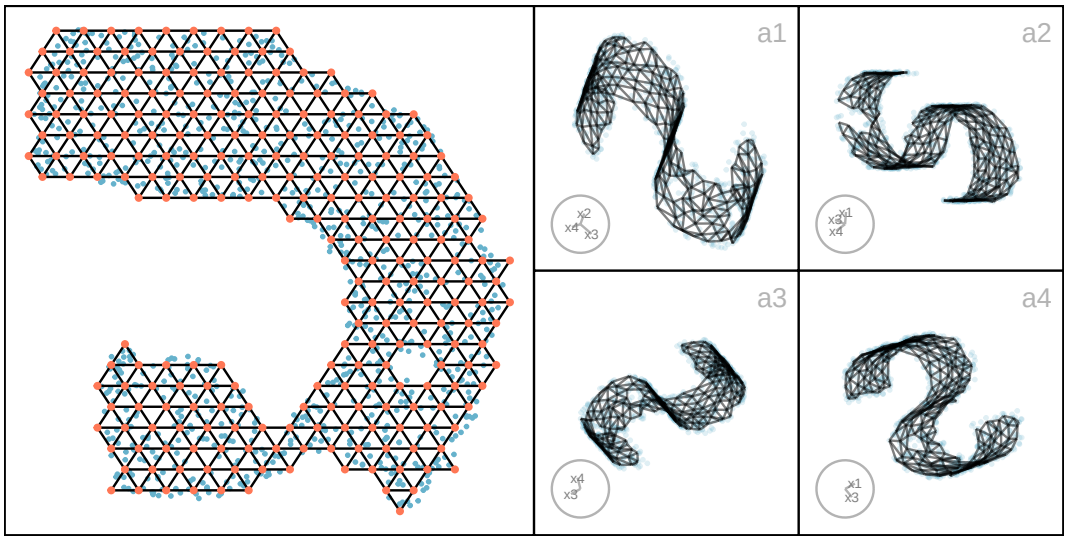

Figure 1: Wireframe model representation of the NLDR layout, lifted and displayed in high-dimensional space. The left panel shows the NLDR layout with a triangular mesh overlay, forming the wireframe structure. This mesh can be lifted into higher dimensions and projected to examine how the geometric structure of the data is preserved. Panels (a1–a4) display different 2-*D* projections of the lifted wireframe, where the underlying curved sheet structure of the data is more clearly visible. The triangulated mesh highlights how local neighborhoods in the layout correspond to relationships in the high-dimensional space, enabling diagnostics of distortion and preservation across dimensions.

The paper is organized as follows. The usage section explains how users can fit a complete

model pipeline from hexagonal binning of a 2-*D* layout to lifting the representation back into *p*-*D* and how the resulting objects can be explored interactively using tour. The next section introduces the implementation of the `quollr` package on CRAN, including a demonstration of the package's key functions and visualization capabilities. In the application section, we illustrate the algorithm's functionality for studying a clustering data structure. Finally, we conclude the paper with a brief summary and discuss potential opportunities for using our algorithm.

2 Usage

The package is available on CRAN, and the development version is available at github.com/JayaniLakshika/quollr.

Our algorithm includes the following steps: (1) scaling the NLDR data, (2) computing configurations of a hexagon grid, (3) binning the data, (4) obtaining the centroids of each bin, (5) indicating neighboring bins with line segments that connect the centroids, and (6) lifting the model into high dimensions. A detailed description of the algorithm can be found in Gamage et al. (2025).

The user needs two inputs: the high-dimensional dataset and the corresponding NLDR data. The high-dimensional data must contain a unique `ID` column, with data columns prefixed by the letter `x` (e.g., `x1`, `x2`, etc.). The NLDR dataset should include embedding coordinates labeled as `emb1` and `emb2`, ensuring one-to-one correspondence with the high-dimensional data through the shared `ID`. The built-in example datasets, `scurve` and `scurve_umap` demonstrates these structures.

To run the entire model pipeline, we can use the `fit_high_model()` function. This function requires: the high-dimensional data (`highd_data`), the embedding data (`nldr_data`), the number of bins along the x-axis (`b1`), the buffer amount as a proportion of the data (`q`), and a benchmark value to identify high-density hexagons (`hd_thresh`).

The function returns an object of class `highd_vis_model` containing the scaled NLDR object (`nldr_scaled_obj`) with three elements: the first is the scaled NLDR data (`scaled_nldr`), and the second and third are the limits of the original NLDR data (`lim1` and `lim2`); the hexagonal object (`hb_obj`), the fitted model in both 2-*D* (`model_2d`), and *p*-*D* (`model_highd`), and triangular mesh (`trimesh_data`).

```
model_obj <- fit_highd_model(
  highd_data = scurve,
  nldr_data = scurve_umap,
  b1 = 21,
  q = 0.1,
  hd_thresh = 0)
```

The resulting model can then be shown in a tour using a two-step process:

```
combined_data <- comb_data_model(
  highd_data = scurve,
  model_highd = model_obj$model_highd,
  model_2d = model_obj$model_2d
)

tour_view <- show_langevitour(
  point_data = combined_data,
  edge_data = model_obj$trimesh_data
)
```

which produces the model and data plot(s) shown in Figure 3.

3 Implementation

The implementation of `quollr` is designed to be efficient, and easy to extend. The package is organized into a series of logical components that reflect the main stages of the workflow: data preprocessing, model fitting, low-density bin removal, prediction, visualization, and interactive exploration (Figure 2). This package structure makes the code easier to maintain and allows new features to be added without changing the existing functionality.

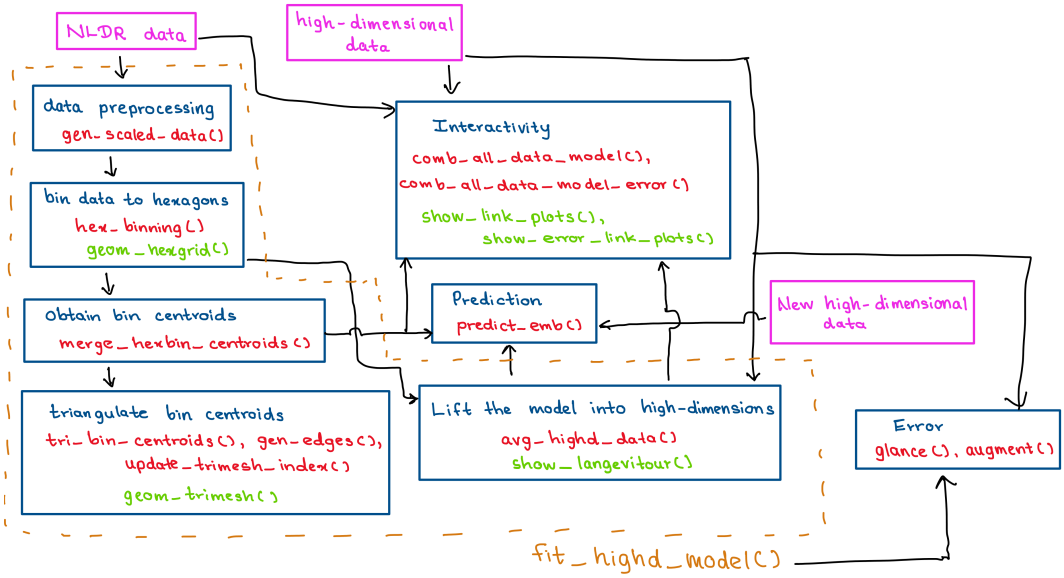

Figure 2: Overview of the 'quollr' workflow and software architecture. The process begins with NLDR and *p*-*D* data inputs, followed by data preprocessing and hexagonal binning. Centroids are computed and triangulated to form the 2-*D* mesh, which is then lifted into the *p*-*D* space. Predictions and error computations are performed on new data, while interactive functions enable dynamic linking between the *p*-*D* and 2-*D* representations.

Software architecture

The package is organized into seven core modules corresponding to stages of the workflow: preprocessing, 2-*D* model construction, lifting the model into *p*-*D*, prediction, error computation, visualizations, and interactivity. Each module performs a distinct task and communicates through data objects.

1. Data preprocessing: The function `gen_scaled_data()` standardizes the embedding data, manage variable naming, and ensure consistent identifiers across high-dimensional and embedded datasets.
2. Construct 2-*D* model: A series of functions `hex_binning()`, `merge_hexbin_centroids()`, `tri_bin_centroids()`, `gen_edges()`, and `update_trimesh_index()` generate the hexagonal grid, compute bin centroids, and connect the triangular mesh that defines local neighborhoods in the 2-*D* space.
3. Lift the model into *p*-*D*: The function `avg_highd_data()` computes the average of the high-dimensional variables for each bin, linking the 2-*D* representation back to the original data space.
4. Prediction: The function `predict_emb()` estimates the embedding of new high-dimensional observations based on the fitted model.
5. Error computation: The `glance()` and `augment()` function summarizes model performance by comparing the predicted and original embeddings.
6. Visualizations: Functions such as `geom_hexgrid()`, `geom_trimesh()`, and `show_langevitour()` provide tools for exploring the fitted models through static and dynamic visualizations.
7. Interactivity: The functions `comb_all_data_model()` and `show_link_plots()` generate interactive linked visualizations that connect the 2-*D* NLDR layout, the corresponding tour view, and the fitted model. Similarly, `comb_all_data_model_error()` and `show_error_link_plots()` integrate the error distribution with the 2-*D* embedding and tour view, enabling interactivity across multiple plots.

Each module is internally independent but connected through data objects (see next section). This modular design simplifies maintenance and allows developers to extend individual components such as substituting different binning approaches, extracting centroids, or visualization tools without altering the overall workflow.

Data objects

The internal data objects follow the tidy data principle: each variable is stored in a column, each observation in a row, and each type of information in its own table. This structure makes the package

easy to use with the `tidyverse` and other visualization tools.

Input objects

- `highd_data`: a tibble containing the original high-dimensional observations with a unique identifier (`ID`) and variable columns prefixed with `x` (e.g., `x1`, `x2`, ...).
- `nldr_data`: a tibble containing two-dimensional embeddings, labeled as `emb1` and `emb2`, matched to the same `ID`s.

Generated objects

- `scaled_nldr_obj`: the output of `gen_scaled_data()`, which rescales the embedding to the range $[0, 1] \times [0, y_{2,\max}]$, where $y_{2,\max} = r_2/r_1$ is the ratio of the embedding ranges. It includes the scaled coordinates (`scaled_nldr`) and the original limits (`lim1`, `lim2`).
- `hex_bin_obj`: the object created by `hex_binning()`, which includes hexagon grid configurations. It includes the binwidth (`a1`), binheight (`a2`), the number of bins along each axis (`b1`, `b2`), the centroids of all hexagons, polygon coordinates, and the assignment of each data point to the hexagon.
- `highd_vis_model`: the main model object returned by `fit_highd_model()`. It stores all components of the fitted model, including the scaled NLDR data (`nldr_scaled_obj`), the hexagonal bin configurations (`hb_obj`), the averaged p-D summaries for each bin (`model_highd`), the corresponding 2-D bin centroids (`model_2d`), and the triangulated mesh connecting neighboring bins (`trimesh_data`).

Computational efficiency and optimization

Several core computations within `quollr` are optimized using compiled C++ code via the `Rcpp` and `RcppArmadillo` packages. While the user interacts with high-level R functions, performance-critical steps such as nearest-neighbor searches (`compute_highd_dist()`), error metrics (`compute_errors()`), 2-D distance calculations (`calc_2d_dist_cpp()`), and generation of hexagon coordinates (`gen_hex_coord_cpp()`) are handled internally in C++. This design provides significant speedups when analyzing large datasets while maintaining a user-friendly R interface. These C++ functions are not exported but are bundled within the package and fully accessible for inspection in the source code.

Pipeline implementation

In this section, we demonstrate the implementation of each step of the pipeline discussed in the Usage section (`fit_highd_model` and `comb_data_model`). Each step can be run independently to ensure flexibility in the modelling aproach.

The algorithm starts by scaling the NLDR data to to the range $[0, 1] \times [0, y_{2,max}]$, where $y_{2,max} = r_2/r_1$ is the ratio of ranges of embedding components. The output includes the scaled NLDR data (`scaled_nldr`) along with the original limits of the embeddings (`lim1`, `lim2`).

```
scurve_umap_obj <- gen_scaled_data(nldr_data = scurve_umap)
```

Hexagon binning

The hexagon binning process builds a complete hexagonal tessellation over the 2-D NLDR embedding and assigns points to bins for downstream modeling. The workflow proceeds through several steps, beginning with selecting an appropriate grid configurations and ending with assigning NLDR points to their corresponding hexagons.

The first step is to determine the configuration of the hexagonal grid. The function `calc_bins_y()` computes the number of bins along the y-axis (`b2`), the hexagon width (`a1`), and the hexagon height (`a2`), based on three inputs: (i) the scaled NLDR object (`nldr_scaled_obj`, containing the scaled embedding and its 2-D limits), (ii) the chosen number of bins along the x-axis (`b1`), and (iii) a buffer proportion (`q`) that expands the grid slightly beyond the observed data range. This buffer ensures that the tessellation fully covers the embedding. By default, `q = 0.1`, but in practice it must be chosen to avoid exceeding the domain of the scaled NLDR data.

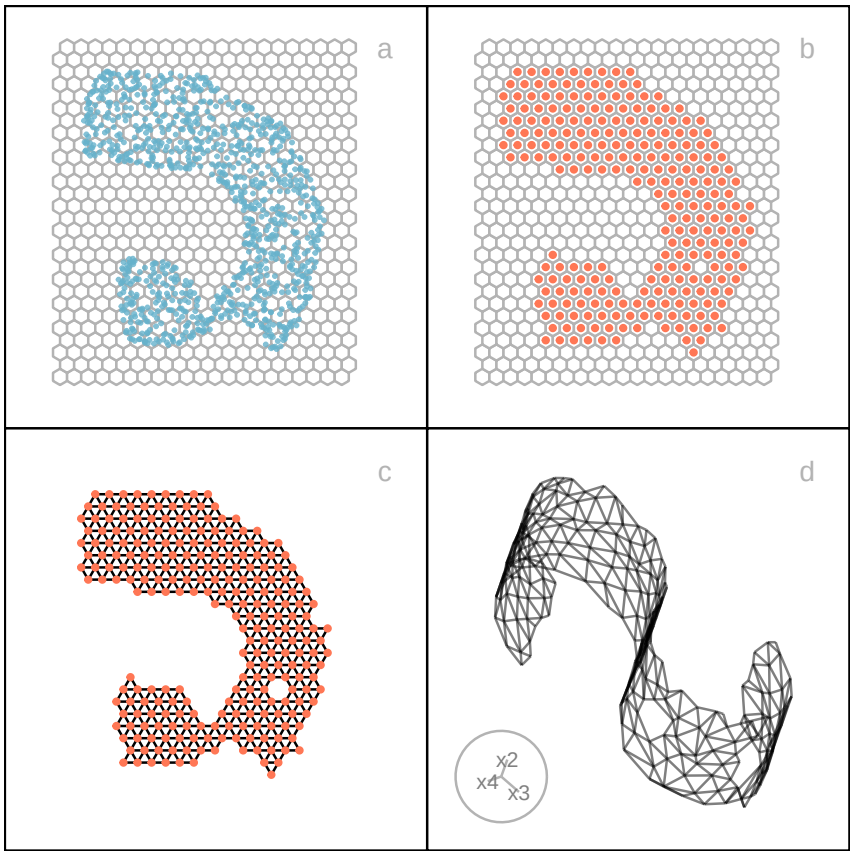

Figure 3: Key steps for constructing the model on the UMAP layout: (a) hexagon bins, (b) bin centroids, (c) triangulated centroids, and (d) lifting the model into high dimensions. The `scurve` data is shown.

```
bin_configs <- calc_bins_y(
  nldr_scaled_obj = scurve_umap_obj,
  b1 = 21,
  q = 0.1)

bin_configs
```

```
> $b2
> [1] 28
>
> $a1
> [1] 0.05869649
>
> $a2
> [1] 0.05083265
```

Once the grid configurations are known, the next step is to construct the full set of hexagon centroids. The function `gen_centroids()` uses the bin configuration and the embedding limits to determine the horizontal and vertical spacing of rows, including the staggered row offsets required for hexagonal tiling. Odd rows place centroids directly above one another, while even rows shift by half the hexagon width to create the interlocking structure. Vertical spacing is determined by $v_s = (\sqrt{3}/2)a_1$, with starting positions defined by the buffer-adjusted limits. This produces a complete grid of potential hexagons covering the 2-D space.

```
all_centroids_df <- gen_centroids(
  nldr_scaled_obj = scurve_umap_obj,
  b1 = 21,
  q = 0.1
)

head(all_centroids_df, 5)
```

```
> # A tibble: 5 x 3
>       h     c_x    c_y
>   <int>   <dbl>  <dbl>
> 1     1 -0.1    -0.116
> 2     2 -0.0413 -0.116
> 3     3  0.0174 -0.116
> 4     4  0.0761 -0.116
> 5     5  0.135  -0.116
```

After determining the centroid locations, each hexagon's polygonal boundary is generated using the `gen_hex_coord()` function, which constructs the six vertices of every hexagon by applying fixed geometric offsets derived from the hexagon width, a_1. Specifically, the horizontal and vertical offsets are defined as $dx = a_1/2$, $dy = a_1/\sqrt{3}$, and $vf = a_1/(2\sqrt{3})$, and these constants are used to position each vertex relative to the centroid, producing the complete set of hexagon boundaries.

These constants define how far each vertex lies from the centroid in the six characteristic directions. For efficiency, the vertex generation is implemented in C++ and returns a tibble listing all polygon vertices, uniquely indexed by hexagons.

```
all_hex_coord <- gen_hex_coord(
  centroids_data = all_centroids_df,
  a1 = bin_configs$a1
)

head(all_hex_coord, 5)
```

```
>   h           x           y
> 1 1 -0.10000000 -0.08179171
> 2 1 -0.12934824 -0.09873593
> 3 1 -0.12934824 -0.13262436
> 4 1 -0.10000000 -0.14956858
> 5 1 -0.07065176 -0.13262436
```

The next step is to assign each NLDR point to its closest hexagon. The `assign_data()` function computes all pairwise Euclidean distances between the 2-D NLDR embedding and the centroid coordinates, identifying the nearest centroid for every observation. Each point is assigned a hexagon index (`h`), producing a version of the embedding annotated with bin membership.

```
umap_hex_id <- assign_data(
  nldr_scaled_obj = scurve_umap_obj,
  centroids_data = all_centroids_df
)

head(umap_hex_id, 5)
```

```
> # A tibble: 5 x 4
>    emb1  emb2    ID     h
>   <dbl> <dbl> <int> <int>
> 1 0.277 0.913     1   427
> 2 0.697 0.538     2   287
> 3 0.779 0.399     3   226
> 4 0.173 0.953     4   446
> 5 0.218 0.983     5   468
```

Finally, the list of points within each hexagon can be extracted using `group_hex_pts()`. This step collapses the point-level assignments into a hexagon-level summary by grouping by `h` and collecting the IDs of all points belonging to that polygon. The result is a tidy representation of the binning structure.

```
pts_df <- group_hex_pts(
  scaled_nldr_hexid = umap_hex_id
)

head(pts_df, 5)
```

```
> # A tibble: 5 x 2
>       h pts_list
>   <int> <list>
> 1    58 <int [4]>
> 2    68 <int [1]>
> 3    69 <int [5]>
> 4    70 <int [6]>
> 5    71 <int [9]>
```

Although each component of the workflow can be run independently, the `hex_binning()` function automates the entire sequence—from grid construction to point assignment—and returns a `hex_bin_obj` containing the bin dimensions (`a1`, `a2`), the full grid geometry (`centroids`, `hex_poly`), bin membership (`data_hb_id`), standardized counts (`std_cts`), the total number of bins (`b`), non-empty bins (`m`), and the list of points per bin (`pts_bins`).

```
hb_obj <- hex_binning(
  nldr_scaled_obj = scurve_umap_obj,
  b1 = 21,
  q = 0.1)
```

Computing the standardized number of points within each hexagon

The `compute_std_counts()` function calculates both the raw and standardized counts of points inside each hexagon.

The function begins by grouping the data by hexagon (`h`) and counting the number of NLDR points falling within each bin. These raw counts are stored as `n_h`. To enable comparisons across bins with varying densities, the function then standardizes these counts by dividing each bin's count by the maximum count across all bins. This yields a standardized bin counts, `w_h`, ranging from 0 to 1.

```
std_df <- compute_std_counts(
  scaled_nldr_h = umap_hex_id
  )

head(std_df, 5)
```

```
> # A tibble: 5 x 3
>       h   n_h   w_h
>   <int> <int> <dbl>
> 1    58     4 0.004
> 2    68     1 0.001
> 3    69     5 0.005
> 4    70     6 0.006
> 5    71     9 0.009
```

Obtaining bin centroids

The `merge_hexbin_centroids()` function combines hexagonal bin coordinates, raw and standardized counts within each hexagons.

This function performs a full join with `centroids_data`, aligning hexagons (`h`) between the two datasets to incorporate both hexagonal bin centroids (`h`) and count metrics. After merging, the function handles missing values in the count columns: any `NA` values in `w_h` or `n_h` are replaced with zeros. This ensures that hexagons with no assigned data points are retained in the output, with zero values for count-related fields. The resulting data contains the full set of hexagon centroids along with associated bin counts (`n_h`) and standardized counts (`w_h`).

```
df_bin_centroids <- merge_hexbin_centroids(
  centroids_data = all_centroids_df,
  counts_data = hb_obj$std_cts
  )

head(df_bin_centroids, 5)
```

```
>   h         c_x        c_y n_h w_h
> 1 1 -0.10000000 -0.1156801   0   0
> 2 2 -0.04130351 -0.1156801   0   0
> 3 3  0.01739298 -0.1156801   0   0
> 4 4  0.07608947 -0.1156801   0   0
> 5 5  0.13478596 -0.1156801   0   0
```

Indicating neighbors by line segments connecting centroids

To represent the neighborhood structure of hexagonal bins in a 2-*D* layout, we employ Delaunay triangulation (Lee and Schachter, 1980; Gebhardt et al., 2024) on the centroids of hexagons.

The `tri_bin_centroids()` function generates a triangulation object from the x and y coordinates of hexagon centroids using the `interp::tri.mesh()` function (Gebhardt et al., 2024).

```
tr_object <- tri_bin_centroids(
  centroids_data = df_bin_centroids
  )
```

The `gen_edges()` function uses this triangulation object to extract line segments between neighboring bins. It constructs a unique set of bin-to-bin connections by identifying the triangle edges and filtering duplicate or reversed links. Each edge is then annotated with its start and end coordinates, and a Euclidean distance between the coordinates.

```
trimesh <- gen_edges(tri_object = tr_object, a1 = hb_obj$a1)

head(trimesh, 5)
```

```
> # A tibble: 5 x 8
>    from    to  x_from  y_from    x_to    y_to from_count to_count
>   <int> <int>   <dbl>   <dbl>   <dbl>   <dbl>      <dbl>    <dbl>
> 1     1     2 -0.1    -0.116  -0.0413 -0.116           0        0
> 2    22    23 -0.0707 -0.0648 -0.0120 -0.0648          0        0
> 3    22    44 -0.0707 -0.0648 -0.0413 -0.0140          0        0
> 4     3    23  0.0174 -0.116  -0.0120 -0.0648          0        0
> 5    44    45 -0.0413 -0.0140  0.0174 -0.0140          0        0
```

The `update_trimesh_index()` function re-indexes the node IDs to ensure that edge identifiers are sequentially numbered and consistent with downstream analysis. This sequential ordering of edges is essential because software such as `langevitour` and `detourr` rely on one-to-one mapping between edges and their corresponding vertices to visualize the mesh.

```
trimesh <- update_trimesh_index(trimesh_data = trimesh)

head(trimesh, 5)
```

```
> # A tibble: 5 x 10
>    from    to  x_from  y_from    x_to    y_to from_count to_count from_reindexed
>   <int> <int>   <dbl>   <dbl>   <dbl>   <dbl>      <dbl>    <dbl>          <int>
> 1     1     2 -0.1    -0.116  -0.0413 -0.116           0        0              1
> 2    22    23 -0.0707 -0.0648 -0.0120 -0.0648          0        0             22
> 3    22    44 -0.0707 -0.0648 -0.0413 -0.0140          0        0             22
> 4     3    23  0.0174 -0.116  -0.0120 -0.0648          0        0              3
> 5    44    45 -0.0413 -0.0140  0.0174 -0.0140          0        0             44
> # i 1 more variable: to_reindexed <int>
```

Identifying and removing low-density hexagons

Not all hexagons contain meaningful information. Some may have very few or no data points due to the sparsity or shape of the underlying structure. Simply removing hexagons with low counts (e.g., fewer than a fixed threshold) can lead to gaps or "holes" in the 2-*D* structure, potentially disrupting the continuity of the representation.

To address this, we propose a more nuanced method that evaluates each hexagon not only based on its own density, but also in the context of its immediate neighbors. The `find_low_dens_hex()` function identifies hexagonal bins with insufficient local support by calculating the average standardized count across their six neighboring bins. If this mean neighborhood density is below a user-defined threshold (e.g., 0.05), the hexagon is flagged for removal.

The `find_low_dens_hex()` function relies on a helper, `compute_mean_density_hex()`, which iterates over all hexagons and computes the average density across neighbors based on their hexagon (`h`) and a defined number of bins along the x-axis (`b1`). The hexagonal layout assumes a fixed grid structure, so neighbor IDs are computed by positional offsets.

```
low_density_hex <- find_low_dens_hex(
  model_2d = df_bin_centroids,
  b1 = 21,
  md_thresh = 0.05
)
```

For simplicity, we remove low-density hexagons using a threshold of 0.

```
df_bin_centroids <- df_bin_centroids |>
  dplyr::filter(n_h > 0)

trimesh <- trimesh |>
  dplyr::filter(from_count > 0,
                to_count > 0)

trimesh <- update_trimesh_index(trimesh)
```

Lifting the model into high dimensions

The final step involves lifting the fitted 2-*D* model into *p*-*D*. This is done by modelling a point in *p*-*D* as the *p*-*D* mean of data points in the 2-*D* centroid. This is performed using the `avg_highd_data()` function, which takes *p*-*D* data (`highd_data`) and embedding data with their corresponding hexagonal bin IDs as inputs (`scaled_nldr_hexid`).

```
df_bin <- avg_highd_data(
  highd_data = scurve,
  scaled_nldr_hexid = hb_obj$data_hb_id
)

head(df_bin, 5)
```

```
> # A tibble: 5 x 8
>       h     x1     x2    x3       x4       x5       x6       x7
>   <int>  <dbl>  <dbl> <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
> 1    58 -0.371 1.91    1.92 -0.00827 0.00189   0.0170   0.00281
> 2    68  0.958 0.0854  1.29  0.00265 0.0171    0.0876  -0.00249
> 3    69  0.855 0.0917  1.51  0.00512 0.000325 -0.0130  -0.00395
> 4    70  0.731 0.129   1.68 -0.00433 0.00211  -0.0356  -0.00240
> 5    71  0.474 0.108   1.88 -0.00260 0.000128  0.00785  0.00170
```

Prediction

The `predict_emb()` function is used to predict a point in a 2-*D* embedding for a new *p*-*D* data point using the fitted model. This function is useful to predict 2-*D* embedding irrespective of the NLDR technique.

In the prediction process, first, the nearest *p*-*D* model point is identified for the new *p*-*D* data point by computing *p*-*D* Euclidean distance. Then, the corresponding 2-*D* bin centroid mapping for the identified *p*-*D* model point is determined. Finally, the coordinates of the identified 2-*D* bin centroid is used as the predicted NLDR embedding for the new *p*-*D* data point.

To accelerate this process, the nearest-neighbor search is implemented in C++ using `Rcpp` via the internal function `compute_highd_dist()`.

```
predict_data <- predict_emb(
  highd_data = scurve,
  model_2d = df_bin_centroids,
  model_highd = df_bin
  )

head(predict_data, 5)
```

```
> # A tibble: 5 x 4
>   pred_emb_1 pred_emb_2    ID pred_h
>        <dbl>      <dbl> <int>  <int>
> 1      0.252      0.901     1    427
> 2      0.692      0.545     2    287
> 3      0.780      0.393     3    226
> 4      0.164      0.952     4    446
> 5      0.193      1.00      5    468
```

It is worth noting that while `predict_emb()` provides a general approach that works across methods, some NLDR techniques have their own built-in prediction mechanisms. For example, UMAP (Konopka, 2023) supports direct prediction of embeddings for new data once a model is fitted.

Compute residuals and hexbin error (HBE)

Hexbin error (HBE) serves as a goodness-of-fit metric for evaluating the high-dimensional to 2-*D* mapping. The `glance()` function computes these summary diagnostics by combining the fitted model returned by `fit_highd_model()` with the *p*-*D* data. After renaming the model output to avoid join conflicts, `predict_emb()` is used to assign each observation to a hexagon in the 2-*D* embedding. The resulting bin assignments are joined with both the model output and the *p*-*D* data, allowing squared and absolute differences between the true and modeled *p*-*D* coordinates to be computed for every dimension. Total absolute error (`Error`) and hexbin error (`HBE`, the root mean squared error) are then obtained using an efficient C++ implementation (`compute_errors()`), and returned in a tidy tibble.

```
glance(
  x = scurve_model_obj,
  highd_data = scurve
)
```

```
> # A tibble: 1 x 2
>   Error   HBE
>   <dbl> <dbl>
> 1  196. 0.116
```

The `augment()` function provides point-level diagnostics by appending predictions and residuals to the original *p*-*D* data. Using the same prediction process as `glance()`, it computes dimension-wise residuals, squared errors, and absolute errors, along with two aggregate measures per observation: the total squared error (`row_wise_total_error`) and the total absolute error (`row_wise_abs_error`). The final output is a tibble containing IDs (`ID`), *p*-*D* data, predicted hexagons (`h`), modeled coordinates, and all residual diagnostics, with one row per observation.

```
model_error <- augment(
  x = scurve_model_obj,
  highd_data = scurve
)
```

Visualizations

The package offers several 2-*D* visualizations (Figure 4), including a full hexagonal grid, a hexagonal grid that matches the data, a full grid based on centroid triangulation, a centroid triangulation grid that aligns with the data, and a triangular mesh for any provided set of points.

The generated *p*-*D* model, overlaid with the data, can also be visualized using `show_langevitour`. Additionally, it features a function for visualizing the 2-*D* projection of the fitted model overlaid on the data, called `plot_proj`.

Furthermore, there are two interactive plots, `show_link_plots` and `show_error_link_plots`, which are designed to help diagnose the model. Each visualization can be generated using its respective function, as described in this section.

Hexagonal grid

The `geom_hexgrid()` function introduces a custom `ggplot2` layer designed for visualizing hexagonal grid on a provided set of bin centroids.

To display the complete grid, users should supply all available bin centroids (Figure 4a).

```
full_hexgrid <- ggplot() +
  geom_hexgrid(
    data = hb_obj$centroids,
    aes(x = c_x, y = c_y)
    )
```

If the goal is to plot only the subset of hexagons that correspond to bins containing data points, then only the centroids associated with those bins should be passed (Figure 4b).

```
data_hexgrid <- ggplot() +
  geom_hexgrid(
    data = df_bin_centroids,
    aes(x = c_x, y = c_y)
    )
```

Triangular mesh

The `geom_trimesh()` function introduces a custom `ggplot2` layer designed for visualizing 2-*D* wireframe on a provided set of bin centroids.

To display the complete wireframe, users should supply all available bin centroids (Figure 4c).

```
full_triangulation_grid <- ggplot() +
  geom_trimesh(
    data = hb_obj$centroids,
    aes(x = c_x, y = c_y)
    )
```

If the goal is to plot only the subset of hexagons that correspond to bins containing data points, then only the centroids associated with those bins should be passed (Figure 4d).

```
data_triangulation_grid <- ggplot() +
  geom_trimesh(
    data = df_bin_centroids,
    aes(x = c_x, y = c_y)
    )
```

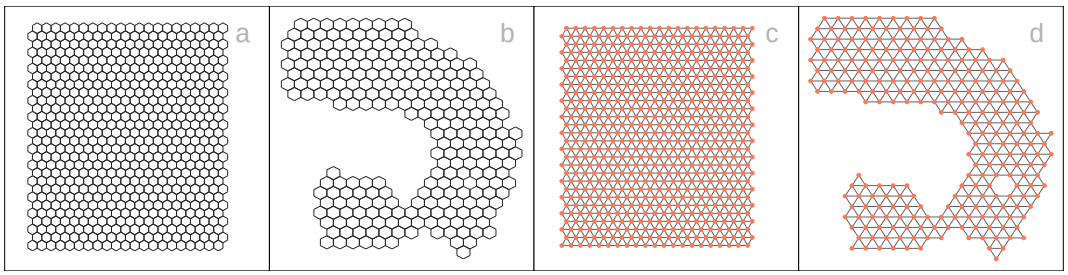

Figure 4: The outputs of `geom_hexgrid` and `geom_trimesh` include: (a) a complete hexagonal grid, (b) a hexagonal grid that corresponds with the data, (c) a full grid based on centroid triangulation, and (d) a centroid triangulation grid that aligns with the data.

p-D model visualization

To visualize how well the *p-D* model captures the underlying structure of the high-dimensional data, we provide a tour of the model in *p-D* using the `show_langevitour()` function (Figure 5). This function renders a dynamic projection of both the high-dimensional data and the model using the `langevitour` R package (Harrison, 2023).

Before plotting, the data needs to be organized into a combined format through the `comb_data_model()` function. This function takes three inputs: `highd_data` (the high-dimensional observations), `model_highd` (high-dimensional summaries for each bin), and `model_2d` (the hexagonal bin centroids of the model). It returns a tidy data frame combining both the data and the model.

In this structure, the `type` variable distinguishes between original observations (`data`) and the bin-averaged model representation (`model`).

```
df_exe <- comb_data_model(
  highd_data = scurve,
  model_highd = df_bin,
  model_2d = df_bin_centroids
  )
```

The `show_langevitour()` function then renders the visualization using the `langevitour` interface, displaying both types of points in a dynamic tour. The `edge_data` input defines connections between neighboring bins (i.e., the hexagonal edges) to visualize the model's structure.

```
show_langevitour(
  point_data = df_exe,
  edge_data = trimesh
  )
```

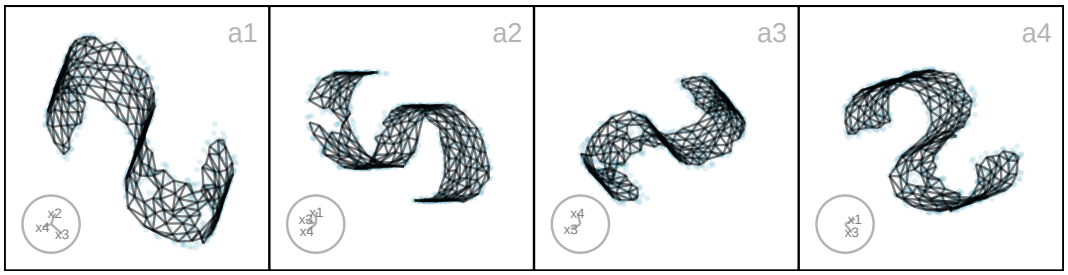

Figure 5: 2-*D* projections of the lifted high-dimensional wireframe model from the `scurve` UMAP layout. Each panel (a1–a4) shows the model (black) overlaid on `scurve` data (blue) in different projections. These views illustrate how the lifted wireframe model captures the structure of the `scurve` data. Regions with sparse or no data in the UMAP layout are also visible in the lifted model.

As an alternative to `langevitour`, users can explore the fitted *p-D* model using the `detourr` (Hart and Wang, 2025) (Figure 6). The combined data object from `comb_data_model()` can be passed directly to the `detour()` function, where `tour_aes()` defines the projection variables and color mapping. The visualization is rendered using `show_scatter()`, which can display both data points and the model's structural edges via the `edges` argument.

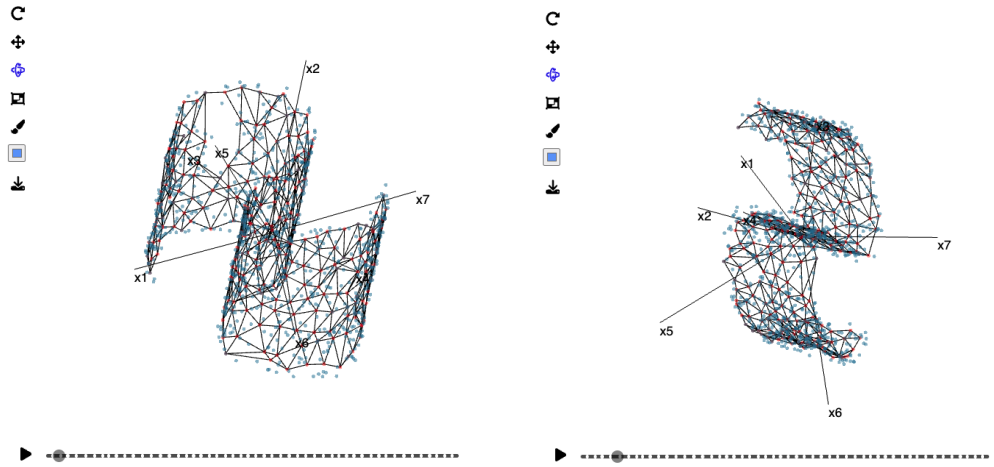

Figure 6: Screenshots of the lifted high-dimensional wireframe model from the `scurve` UMAP layout using `detourr`. Regions with sparse or no data in the UMAP layout are also visible in the lifted model.

```
detour(
  df_exe,
  tour_aes(
    projection = starts_with("x"),
    colour = type
  )
) |>
  tour_path(grand_tour(2),
                    max_bases=50, fps = 60) |>
  show_scatter(axes = TRUE, size = 1.5, alpha = 0.5,
               edges = as.matrix(trimesh[, c("from_reindexed", "to_reindexed")]),
               palette = c("#66B2CC", "#FF7755"),
               width = "600px", height = "600px")
```

In the resulting interactive visualization, blue points represent the high-dimensional data, orange points represent the model centroids from each bin, and the lines between model points reflect the 2-*D* wireframe structure mapped to high-dimensional space.

Linked plots

Two types of interactively linked plots can be generated to assess the model fits everywhere, or better in some subspaces, or completely mismatch the data. The plots are linked using `crosstalk`, which allows interactive brushing: selecting or brushing points in one plot automatically highlights the corresponding points in the other linked views.

First, the function `show_link_plots()` provides linking a 2-*D* NLDR layout and a tour (via `langevitour`) of the model overlaid the data (Figure 7).

The `point_data` for `show_link_plots()` can be prepared using the `comb_all_data_model()` function. This function combines the high-dimensional data (`highd_data`), the NLDR data (`nldr_data`), and the bin-averaged high-dimensional model representation (`model_highd`) aligned to the 2-*D* bin layout (`model_2d`). This combined dataset includes both the original observations and the bin-level model averages, labeled with a `type` variable for distinguishing between them. Also, the `show_link_plots()` function takes `edge_data`, which defines connections between neighboring bins.

```
df_exe <- comb_all_data_model(
  highd_data = scurve,
  nldr_data = scurve_umap,
  model_highd = df_bin,
  model_2d = df_bin_centroids
  )

nldrdt_link <- show_link_plots(
  point_data = df_exe,
  edge_data = trimesh,
```

```
  point_colour = clr_choice
  )

nldrdt_link <- crosstalk::bscols(
    htmltools::div(
        style = "display: grid; grid-template-columns: 1fr 1fr;
    gap: 0px; align-items: start; justify-items: center; margin: 0; padding: 0;
    height: 380px; width: 500px",
        nldrdt_link
    ),
    device = "xs"
)

class(nldrdt_link) <- c(class(nldrdt_link), "htmlwidget")

nldrdt_link
```

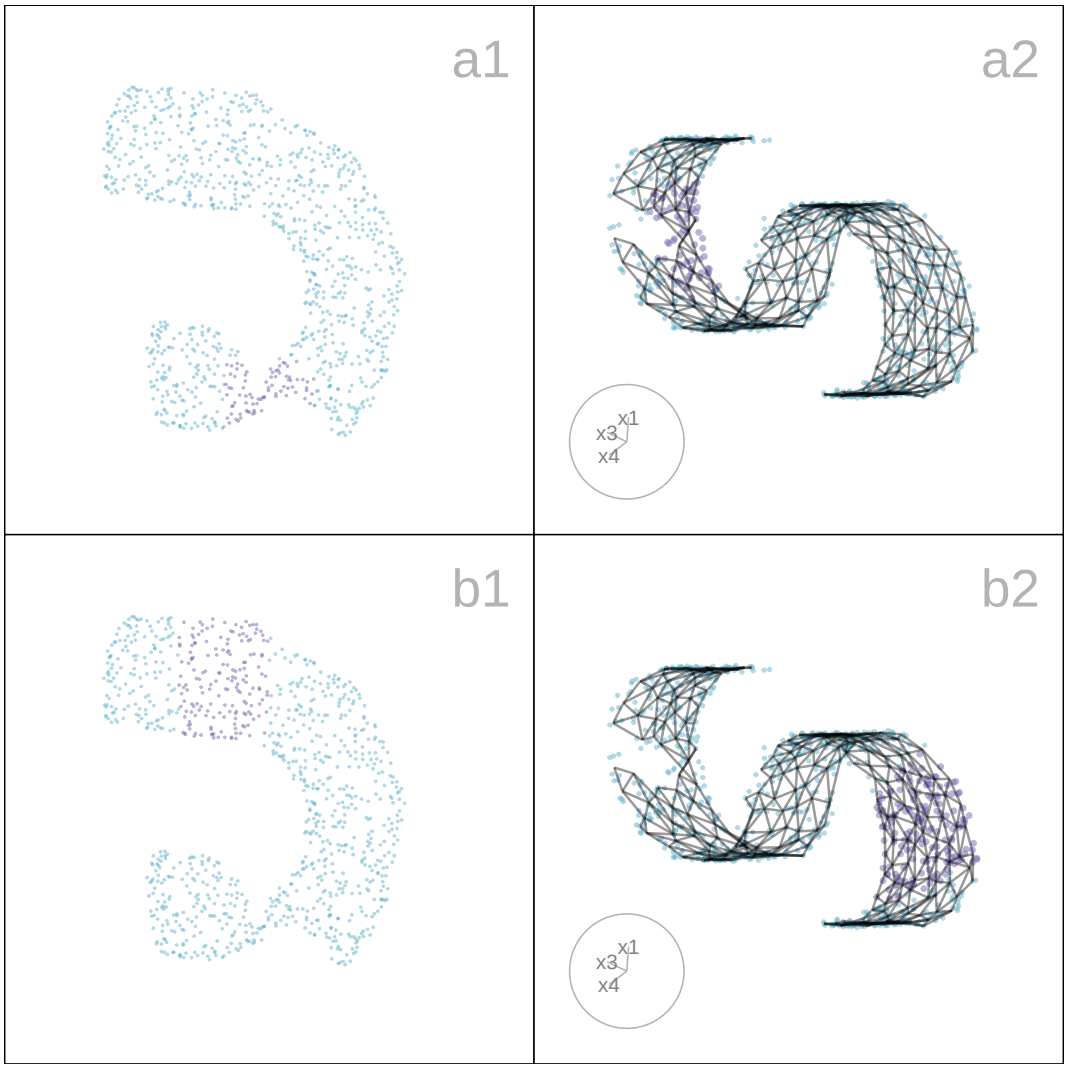

Figure 7: Exploring the correspondence between UMAP layout and `scurve` structure in 7-*D*. Two sets of plots are interactively linked: UMAP layout (a1, b1) and projection of 7-*D* model and data (a2, b2). The purple points indicate the selected subsets, which differ between rows. In (a1), the lower bridge of the `scurve` is highlighted, which corresponds in (a2) to points spanning across both arms of the high-dimensional structure. In (b1), a different region near the upper arm of the `scurve` is selected, and in (b2) these points map onto one side of the curved manifold in 7-*D* projection. While the UMAP layout suggests distinct local clusters, the linked tour views reveal how these selections trace continuous structures in the 7-*D* space, highlighting distortions introduced by UMAP.

The function `show_error_link_plots()` generates three side-by-side, interactively linked plots;

a error distribution, a 2-*D* NLDR layout, and a tour (via `langevitour`) of the model overlaid the data (Figure 8). The function takes the output from `comb_all_data_model_error()` (`point_data`) and `edge_data` which defines connections between neighboring bins.

The `point_data` can be generated using the `comb_all_data_model_error()` function. The function requires several arguments: points data which contain high-dimensional data (`highd_data`), NLDR data (`nldr_data`), high-dimensional model data (`model_highd`), 2-*D* model data (`model_2d`), and model error (`error_data`). This combined dataset includes both the original observations and the bin-level model averages, labeled with a `type` variable for distinguishing between them.

```
df_exe <- comb_all_data_model_error(
  highd_data = scurve,
  nldr_data = scurve_umap,
  model_highd = df_bin,
  model_2d = df_bin_centroids,
  error_data = model_error
  )

errornldrdt_link <- show_error_link_plots(
  point_data = df_exe,
  edge_data = trimesh,
  point_colour = clr_choice
)

class(errornldrdt_link) <- c(class(errornldrdt_link), "htmlwidget")

errornldrdt_link
```

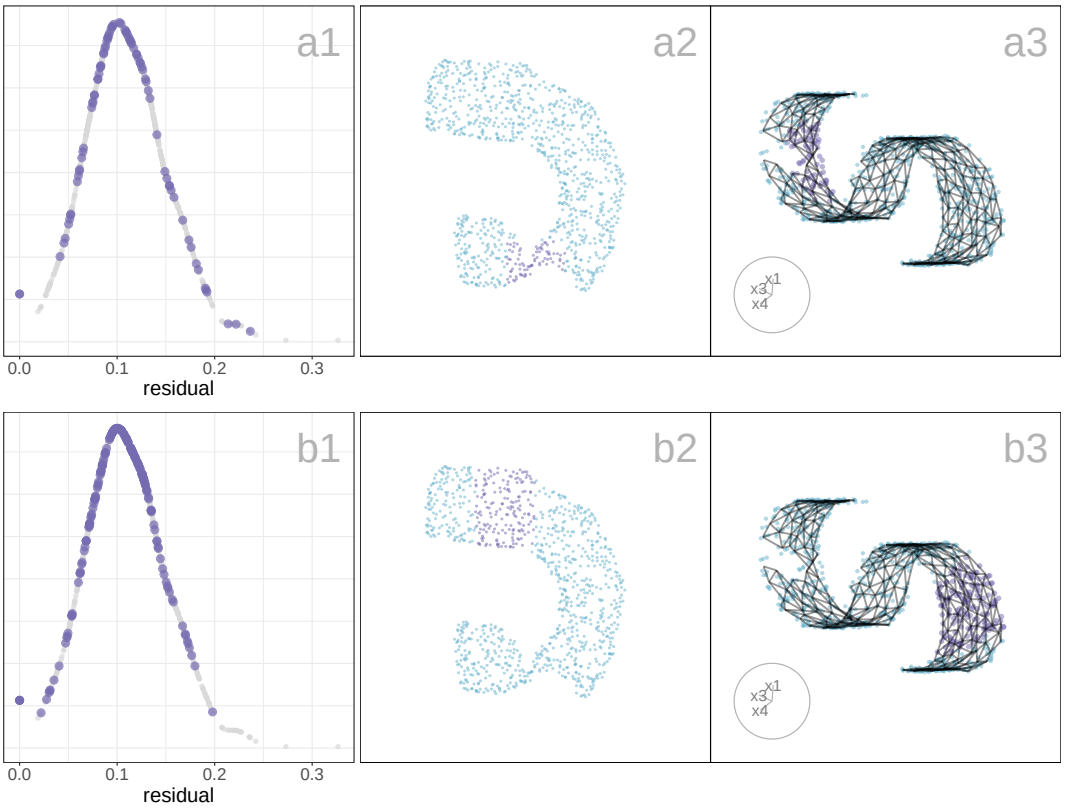

Figure 8: Exploring residuals in relation to UMAP layouts using a 7-*D* `scurve` model. Three views are linked: distribution of residuals (a1, b1), UMAP layout (a2, b2), and projection of the 7-*D* model with data (a3, b3). The purple points highlight selected subsets of the data, which differ across rows. In the top row (a1–a3), points with higher residuals (a1) are selected, corresponding to the sparse bridging region in the UMAP layout (a2) and the less dense end of the `scurve` in the high-dimensional projection (a3). In the bottom row (b1–b3), points with lower residuals (b1) are highlighted, which map to one side of the dense region in the NLDR layout (b2) and to a thicker band of the `scurve` in the projection (b3). This comparison illustrates how residuals can help diagnose distortions in UMAP, with high-residual points often concentrated in sparse or stretched regions of the structure.

4 Application

Single-cell RNA sequencing (scRNA-seq) is a popular and powerful technology that allows you to profile the whole transcriptome of a large number of individual cells (Andrews et al., 2021).

Clustering of single-cell data is used to identify groups of cells with similar expression profiles. NLDR often used to summarize the discovered clusters, and help to understand the results. The purpose of this example is to *illustrate how to use our method to help decide on an appropriate NLDR layout that accurately represents the data*.

Limb muscle cells of mice in et al. (2018) are examined. There are 1067 single cells, with 14997 gene expressions. Following their pre-processing, different NLDR methods were performed using ten principal components. Figure 9 (a) is the reproduction of the published plot. This was generated using tSNE with perplexity $= 30$, the default hyper-parameters. The question is whether this accurately represents the cluster structure in the data. Note that the cluster variable is not used to produce the 2-*D* layout.

We illustrate how to use `quollr` to assess whether this is a reasonable layout. Figure 9 shows five alternative layouts, and the HBE plot summarizing the resulting model fits. Layout b is produce by UMAP (neighbors $= 5,$ minimum distance $= 0.1$); layout c was produced by PHATE (knn $= 5$); layout d was produced by TriMAP (number of inliers $= 12,$ outliers $= 4,$ random $= 3$); layout e was produced by by PaCMAP (neighbors $= 10,$ init = "random", MN-ratio $= 0.5,$ FP-ratio $= 2$); layout f was produced by tSNE (perplexity $= 15$).

The HBE plot indicates that the two tSNE layouts outperform all the other methods across a range of binwidths, but that the result with perplexity of 15 outperforms the other. There are small visual difference between the two layouts. Both support that there are $5 - 6$ clusters. Layout a has slightly more space between clusters. Layout d three small clusters at the top whereas layout f has only two, and another smaller one at the bottom.

```
design <- gen_design(n_right = 6, ncol_right = 2)

plot_hbe_layouts(plots = list(error_plot_limb, nldr1,
                              nldr2, nldr3, nldr4,
                              nldr5, nldr6), design = design)
```

Figure 10 show how to examine the resulting models of the layouts overlaid on the data in high dimensions. Point color represents the cluster reported in the published paper. In each case the best model is produced using binwidth $= 0.06$. A binwidth of 0.06 is used because it is small enough to show local structure and differences between clusters, but large enough to avoid breaking the layout into too many small pieces that would make the plots hard to interpret. The plots in these figures are best understand using small steps.

1. Examine the model in the data space for each, by looking at the tour views. In each case, the clustering doesn't quite match the separations in the data, and both models help see this. For example, the orange cluster (1) should probably be split into more than one cluster because both models show large stretched lines connecting a small group far from the remaining orange points.
2. Because 6 clusters are hard to examine together, use the menu to select just one cluster to view at at time. Selecting just cluster 1 might help you see the explanation above, that a small group is quite separate from the main group. This suggests both the layouts and the clustering that groups these together might be wrong. Now change to focus on cluster 5 (yellow). This group is a fairly large sparse cluster but it is separated from the other points. Both layouts are right in separating these points from the others.
3. To examine where the layouts differ, examine clusters 4 (green) and 6 (cream), by selecting just these two. Layout a has them close, but layout f has them far apart. (It might also help to include cluster 5 here because layout a has this group close to the cluster 4 also). In the tour view, we can see that they three clusters are separated clusters in different directions away from the most of the other points. They are both correct in this. It may have been better to place them all in different corners of the layout, but they have preserved the most important aspect that they are separated clusters. That they are all close together in layout a could be incorrectly interpreted as close in high-dimensions though.

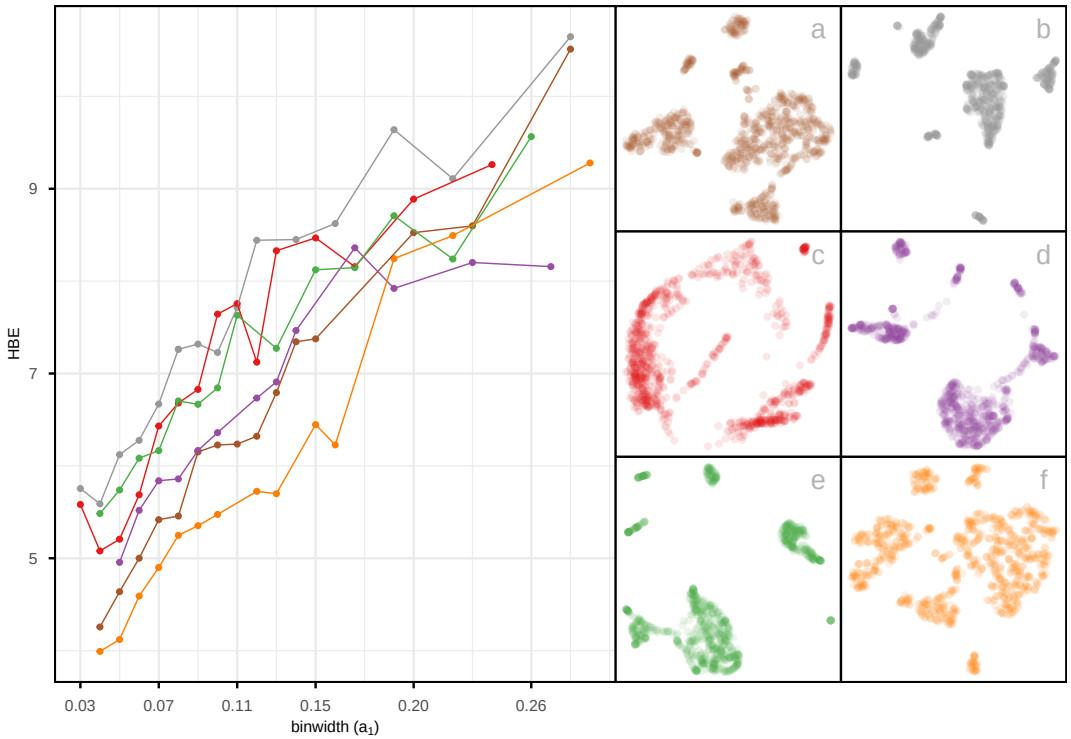

Figure 9: Assessing which of the 6 NLDR layouts on the limb muscle data is the better representation using HBE for varying binwidth (a_1). Colour used for the lines and points in the left plot and in the scatterplots represents NLDR layout (a-f). Layout d performs well at large binwidth (where the binwidth is not enough to capture the data structure) and poorly as the bin width decreases. Layout f is the best choice.

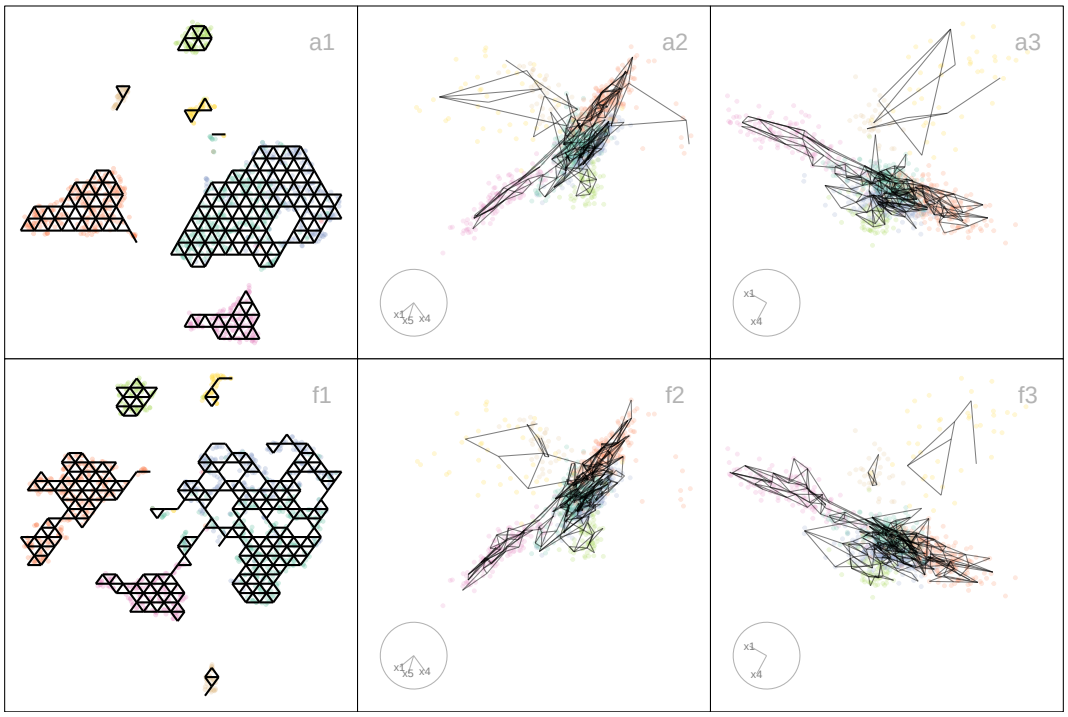

Figure 10: Compare the published 2-*D* layout (Figure 9a) and the 2-*D* layout selected (Figure 9f) by HBE plot (Figure 9) from the tSNE, UMAP, PHATE, TriMAP, and PaCMAP with different hyper-parameters. The Limb muscle data ($n = 1067$) has seven close different shaped clusters in 10-*D*.

5 Discussion

The `quollr` package introduces a new framework for interpreting NLDR outputs by fitting a geometric wireframe model in 2-*D* and lifting it into high-dimensional space. This lifted model provides a

direct way to assess how well a 2-*D* layout, produced by methods such as tSNE, UMAP, PHATE, TriMAP, or PaCMAP, preserves the structure of the original high-dimensional data. The approach offers both numerical and visual diagnostics to support the selection of NLDR methods and tuning hyper-parameters that produce the most accurate 2-*D* representations.

In contrast to the common practice of visually inspecting scatterplots for clusters or patterns, `quollr` provides a quantitative route for evaluation. It enables the computation of HBE and residuals between the original high-dimensional data and the lifted model, offering interpretable diagnostics. These diagnostics are complemented by interactive linked plots and high-dimensional dynamic visualizations using the `langevitour` package, allowing users to inspect where the model fits well and where it does not.

To support efficient computation, particularly for large-scale datasets, several core functions in `quollr` are implemented in C++ using `Rcpp` and `RcppArmadillo`. These include functions for computing Euclidean distances in high-dimensional and 2-*D* space, identifying nearest centroids, calculating residual errors, and generating polygonal coordinates of hexagons. For instance, `compute_highd_dist()` accelerates nearest neighbor lookup in high-dimensional space, `compute_errors()` calculates HBE and total absolute error efficiently, and `calc_2d_dist_cpp()` speeds up distance calculations in 2-*D*. Additionally, `gen_hex_coord_cpp()` constructs the coordinates for hexagonal bins based on their centroids with minimal overhead. These optimizations result in substantial performance gains compared to native R implementations, making the package responsive even when used in interactive contexts or on large datasets such as single-cell transcriptomic profiles.

The modular structure of the package is designed to support both flexibility and reproducibility. Users can access individual functions to control each step of the pipeline such as scaling, binning, and triangulation or use the main function `fit_highd_model()` for end-to-end model construction. The diagnostics can be used not only to compare NLDR methods but also to tune binning parameters, assess layout stability, and detect local distortions in the embedding.

There are several avenues for future development. While hexagonal binning provides a regular structure conducive to modeling, alternative spatial discretizations (e.g., adaptive binning or density-aware tessellations) could be explored to better capture varying data densities. Expanding support for additional distance metrics in the lifting and prediction steps may improve performance across different domains. Additionally, statistical inference tools could be introduced to assess the stability and robustness of the fitted model, which would enhance interpretability and confidence in the outcomes. Currently, linked plot functionality is limited: selecting a cluster in the `langevitour` controls does not yet highlight the corresponding cluster in the 2-*D* layout. Implementing this feature in the future would allow users to clearly explore cluster-specific relationships between the `langevitour` and 2-*D* layout.

6 Acknowledgements

The source code for reproducing this paper can be found at: github.com/JayaniLakshika/paper-quollr. This article is created using **knitr** (Xie, 2015) and **rmarkdown** (Xie et al., 2018) in R with the `rjtools::rjournal_article` template. These `R` packages were used for this work: `cli` (Csárdi, 2025), `dplyr` (Wickham, 2023), `ggplot2` (Wickham, 2016), `interp` (>= 1.1-6) (Gebhardt et al., 2024), `langevitour` (Harrison, 2023), `detourr` (Hart and Wang, 2025), `proxy`(Meyer and Buchta, 2022), `stats` (R Core Team, 2025), `tibble` (Müller and Wickham, 2023), `tidyselect` (Henry and Wickham, 2024), `crosstalk` (Cheng and Sievert, 2023), `plotly` (Sievert, 2020), `htmltools` (Cheng et al., 2024), `kableExtra` (Zhu, 2024), `patchwork` (Pedersen, 2024), and `readr` (Wickham et al., 2024).

Bibliography

E. Amid and M. K. Warmuth. Trimap: Large-scale dimensionality reduction using triplets. *ArXiv*, abs/1910.00204, 2019. URL https://api.semanticscholar.org/CorpusID:203610264. [p1]

T. S. Andrews, V. Y. Kiselev, D. McCarthy, and M. Hemberg. Tutorial: guidelines for the computational analysis of single-cell rna sequencing data. *Nature Protocols*, 16(1):1–9, 2021. URL https://doi.org/10.1038/s41596-020-00409-w. [p16]

J. Cheng and C. Sievert. *crosstalk: Inter-Widget Interactivity for HTML Widgets*, 2023. URL https://CRAN.R-project.org/package=crosstalk. R package version 1.2.1. [p18]

J. Cheng, C. Sievert, B. Schloerke, W. Chang, Y. Xie, and J. Allen. *htmltools: Tools for HTML*, 2024. URL https://CRAN.R-project.org/package=htmltools. R package version 0.5.8.1. [p18]

G. Csárdi. *cli: Helpers for Developing Command Line Interfaces*, 2025. URL https://CRAN.R-project.org/package=cli. R package version 3.6.4. [p18]

T. M. C. et al. Single-cell transcriptomics of 20 mouse organs creates a tabula muris. *Nature*, 562(7727): 367–372, 2018. [p16]

J. P. Gamage, D. Cook, P. Harrison, M. Lydeamore, and T. S. Talagala. Choosing better nldr layouts by evaluating the model in the high-dimensional data space, 2025. URL https://arxiv.org/abs/2506.22051. [p2]

A. Gebhardt, R. Bivand, and D. Sinclair. *interp: Interpolation Methods*, 2024. URL https://CRAN.R-project.org/package=interp. R package version 1.1-6. [p8, 18]

P. Harrison. langevitour: Smooth interactive touring of high dimensions, demonstrated with scrna-seq data. *The R Journal*, 15(2):206–219, 2023. doi: 10.32614/RJ-2023-046. [p12, 18]

C. Hart and E. Wang. *detourr: Portable and Performant Tour Animations*, 2025. URL https://CRAN.R-project.org/package=detourr. R package version 0.2.0. [p12, 18]

L. Henry and H. Wickham. *tidyselect: Select from a Set of Strings*, 2024. URL https://CRAN.R-project.org/package=tidyselect. R package version 1.2.1. [p18]

T. Konopka. *umap: Uniform Manifold Approximation and Projection*, 2023. URL https://CRAN.R-project.org/package=umap. R package version 0.2.10.0. [p10]

D. T. Lee and B. J. Schachter. Two algorithms for constructing a delaunay triangulation. *International Journal of Computer & Information Sciences*, 9(3):219–242, 1980. URL https://doi.org/10.1007/BF00977785. [p8]

L. V. D. Maaten and G. E. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9: 2579–2605, 2008. [p1]

L. McInnes, J. Healy, N. Saul, and L. Großberger. UMAP: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861, 2018. URL https://doi.org/10.21105/joss.00861. [p1]

D. Meyer and C. Buchta. *proxy: Distance and Similarity Measures*, 2022. URL https://CRAN.R-project.org/package=proxy. R package version 0.4-27. [p18]

K. R. Moon, D. van Dijk, Z. Wang, S. A. Gigante, D. B. Burkhardt, W. S. Chen, K. Yim, A. van den Elzen, M. J. Hirn, R. R. Coifman, N. B. Ivanova, G. Wolf, and S. Krishnaswamy. Visualizing structure and transitions in high-dimensional biological data. *Nature Biotechnology*, 37:1482–1492, 2019. [p1]

K. Müller and H. Wickham. *tibble: Simple Data Frames*, 2023. URL https://CRAN.R-project.org/package=tibble. R package version 3.2.1. [p18]

T. L. Pedersen. *patchwork: The Composer of Plots*, 2024. URL https://CRAN.R-project.org/package=patchwork. R package version 1.3.0. [p18]

R Core Team. *R: A Language and Environment for Statistical Computing*, 2025. URL https://www.R-project.org/. [p18]

C. Sievert. *Interactive Web-Based Data Visualization with R, plotly, and shiny*. Chapman and Hall/CRC, 2020. URL https://plotly-r.com. [p18]

Y. Wang, H. Huang, C. Rudin, and Y. Shaposhnik. Understanding how dimension reduction tools work: An empirical approach to deciphering t-sne, umap, trimap, and pacmap for data visualization. *Journal of Machine Learning Research*, 22(201):1–73, 2021. URL http://jmlr.org/papers/v22/20-1061.html. [p1]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. URL https://ggplot2.tidyverse.org. [p18]

H. Wickham. *conflicted: An Alternative Conflict Resolution Strategy*, 2023. URL https://CRAN.R-project.org/package=conflicted. R package version 1.2.0. [p18]

H. Wickham, J. Hester, and J. Bryan. *readr: Read Rectangular Text Data*, 2024. URL https://CRAN.R-project.org/package=readr. R package version 2.1.5. [p18]

Y. Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, 2nd edition, 2015. URL https://yihui.name/knitr/. [p18]

Y. Xie, J. Allaire, and G. Grolemund. *R Markdown: The Definitive Guide*. Chapman and Hall/CRC, 2018. URL https://bookdown.org/yihui/rmarkdown. [p18]

H. Zhu. *kableExtra: Construct Complex Table with 'kable' and Pipe Syntax*, 2024. URL https://CRAN.R-project.org/package=kableExtra. R package version 1.4.0. [p18]

Jayani P. Gamage
Monash University
Department of Econometrics and Business Statistics, VIC 3800 Australia
https://jayanilakshika.netlify.app/
ORCiD: 0000-0002-6265-6481
jayani.piyadigamage@monash.edu

Dianne Cook
Monash University
Department of Econometrics and Business Statistics, VIC 3800 Australia
http://www.dicook.org/
ORCiD: 0000-0002-3813-7155
dicook@monash.edu

Paul Harrison
Monash University
MGBP, BDInstitute, VIC 3800 Australia
ORCiD: 0000-0002-3980-268X
paul.harrison@monash.edu

Michael Lydeamore
Monash University
Department of Econometrics and Business Statistics, VIC 3800 Australia
ORCiD: 0000-0001-6515-827X
michael.lydeamore@monash.edu

Thiyanga S. Talagala
University of Sri Jayewardenepura
Department of Statistics, Gangodawila, Nugegoda 10100 Sri Lanka
https://thiyanga.netlify.app/
ORCiD: 0000-0002-0656-9789
ttalagala@sjp.ac.lk