scDEcrypter: Detecting latent viral states in scRNA-seq

Luer Zhong

2026-02-24

Introduction

Detecting infected cells in viral scRNA-seq data is challenging because viral transcripts are often sparse or undetected, leaving most infection labels unknown and limiting downstream differential expression analyses. scDEcrypter is a statistical framework we developed to address this problem by modeling infection status, and optionally an additional partitioning variables such as cell type. The method uses a penalized multi-way mixture model, anchored by a small set of confidently labeled cells, to recover latent infection states with high accuracy even when viral reads are extremely limited. To avoid double-dipping, scDEcrypter employs a data-splitting strategy, using one subset of cells for parameter estimation and another independent subset for differential expression testing. A fast approximate likelihood ratio test is then used to identify infection-associated genes within each cell type. scDEcrypter can improve power, robustness, and biological interpretability.

Run scDEcrypter

Install the package

Before analysis can be performed, scDEcrypter must be installed. The easiest way to install scDEcrypter is through GitHub:

```
library(devtools)
# devtools::install_github("https://github.com/bachergroup/scDEcrypter")

library(scDEcrypter)
```

Required inputs

The input data should be a Seurat object. For this walk-through, we will analyze the a small subset of cells from the SARS-CoV-2 dataset from Ravindra NG, et al. **Single-cell longitudinal analysis of SARS-CoV-2 infection in human airway epithelium identifies target cells, alterations in gene expression, and cell state changes**. PLoS Biol. 2021;19(3):e3001143. For the purpose of this vignette, we have 2,000 cells randomly subset from the MOCK (uninfected control sample) and DPI3 (three days post-infection).

```
set.seed(120)
data("seu_sub")

table(seu_sub$ctype, seu_sub$Sample)
```

```
##
##                      DPI3 MOCK
##   Basal cells         557  348
##   BC/Club              47   72
```

```
##   Ciliated cells        310  254
##   Club cells            183  194
##   Goblet cells            1    2
##   Ionocytes              15    7
##   Neuroendocrine cells    3    2
##   Tuft cells              4    1
```

Define the partitioning variable

scDEcrypter requires an additional cell-level label, which we refer to as a partitioning variable. This variable defines groups of cells that are expected to differ in their baseline expression patterns, and it helps the model separate biological heterogeneity unrelated to viral infection status. Importantly, the partitioning variable is flexible and does not need to be cell type. It can represent any categorical cell-level attribute that the user deems relevant.

For the example here, we use eight epithelial cell-type categories as the partitioning variable. Users may adapt or replace this mapping entirely depending on the structure of their own dataset.

The partitioning variable must be numerically coded.

```
seu_sub$C.preLabel <- as.integer(factor(seu_sub$ctype))
```

Define the viral status variable

In scRNA-seq viral infection datasets, each cell's infection status is not fully observed. scDEcrypter is explicitly designed to model this partially latent structure. These confidently labeled cells serve as anchors for the mixture model, while the remaining cells are treated as unknown and their infection status is inferred probabilistically during model fitting.

As input data, scDEcrypter requires requires a cell-level viral status variable that indicates whether each cell has a confidently known status or is unknown. Note that our model is not restricted to having just two infection states, and the user may specify this directly in the next step. In general, we recommend the following pre-labeling approach:

(1). cells with strong viral read evidence and can be confidently labeled as infected (e.g. reads greater than 1),

(2). cells from unexposed or healthy samples can be confidently labeled as uninfected,

(3). all other cells have an unknown status, such that the absence of viral reads does not guarantee that the cell is truly uninfected (e.g., low viral load or bystander cells).

To provide this information to scDEcrypter numerically, we recommend:

(1). assign 1 to confidently infected cells,

(2). assign 2 to confidently uninfected cells,

(3). leave all other cells as NA to allow scDEcrypter to infer their latent infection status.

A typical setup might look like below. Cells from the Healthy patients are pre-labeled as Uninfected and cells that were confidently deemed Infected based on observed viral mRNA counts are pre-labelled as Infected.

```
V.obs <- rep(NA, ncol(seu_sub))
to_label_I <- which(seu_sub$Total_COV >= 40)
to_label_U <- which(seu_sub$Sample == "MOCK")
V.obs[to_label_I] <- 1
V.obs[to_label_U] <- 2

seu_sub$V.preLabel <- V.obs
```

Define the number of infection states and partitioning variable classes

scDEcrypter models gene expression using a multiway mixture model, where each cell belongs to a latent combination of a viral status class (e.g., infected, uninfected), and a partitioning variable class (e.g., cell type, biological group, or any other cell-level label).

To fit this model, users must additionally specify:

v_star: the total number of viral status categories the model should consider,

c_star: the total number of categories for the chosen partitioning variable.

These values define the full set of latent and observed combinations that scDEcrypter will learn.

In many viral scRNA-seq datasets, infection status is not simply "infected vs uninfected." Cells may exhibit transcriptional profiles consistent with an intermediate 'bystander' state, where they are not infected but respond to nearby infected cells. Thus, we may want to allow for the possibility of three infection states. This allows scDEcrypter to infer these three viral states separately and perform differential expression comparisons between them.

The partitioning variable (e.g., cell type) can also be partially latent. Users must specify the total number of categories the model should expect for that variable as well. For this dataset, we stick to the number of annotated cell-types and it is fully labeled.

```
v_star <- 3
c_star <- 8
```

Run the pre-processing function

To accurately recover latent infection states and perform unbiased differential expression testing, scDEcrypter separates the dataset into two independent subsets:

A generation set — used only for parameter estimation in the penalized multiway mixture model;

A test set — used only for downstream inference and likelihood-ratio testing.

This split ensures that the same data are not used for both estimating mixture parameters and testing hypotheses, thereby avoiding "double dipping," a key principle emphasized in the scDEcrypter paper.

The function preprocess_scDEcrypter() automates all required pre-processing steps. The function returns a Seurat object with the VST normalized data in the data.Generation and data.Test. The metadata also includes a column named Index_Split that indicated the set each cell belongs to. The parameter train_frac controls the proportion of splitting between generation and test sets (stratified by known infection and partitioning labels). The parameters c_obs_var and v_obs_var specify the meta-data column containing the pre-labels. The default names are C.preLabel and V.preLabel as in the steps above. Users may specify their own columns if named differently.

Variance stabilization is an essential part of the scDEcrypter workflow as the underlying mixture model assumes approximately Gaussian stabilized expression values. Through simulations and applications we found "shifted_log" performed consistently well.

```
seu_sub.processed <- preprocess_scDEcrypter(Data = seu_sub,
                                            train_frac = .5)
```

```
## Splitting data into generation and test sets...

## Applying VST to generation set...

## Warning: multiple methods tables found for 'union'

## Warning: multiple methods tables found for 'intersect'

## Warning: multiple methods tables found for 'setdiff'
```

```
## Applying VST to test set...
```

scDEcrypter uses a penalized multi-way mixture model, which is computationally efficient and statistically stable when trained on a compact set of informative genes. Therefore, HVG selection is performed only on the generation set, not the full dataset. This prevents information leakage from the training set and preserves the integrity of the data-splitting framework described in the manuscript.

In order to select genes driving a truly infected versus bystander or uninfected signal, we recommend a cell-type–aware HVG selection procedure that considers cell type and sample status. Users may customize the HVG selection method, as long as HVGs are selected only from the generation set, and the resulting HVG set is passed into the model-fitting step. This ensures that downstream inference remains valid and independent from the model training process.

We suggest anywhere between 100-200 highly variable genes selected in total once aggregated across cell-types.

```
seurat_train <- subset(seu_sub.processed, Index_Split == "Generation")

top_genes.HVG <- get_top_genes(
                    seurat_obj = subset(seurat_train, Sample == "DPI3"),
                    partition_colname = "C.preLabel",
                    n_genes_per_group = 50)

length(top_genes.HVG)
```

```
## [1] 179
```

Cross validation to select tuning parameter

scDEcrypter employs a penalized mixture modeling framework, where the penalty parameter lambda controls the amount of regularization applied to the model's mean parameters. Choosing an appropriate lambda is essential for balancing model flexibility with stability, and for recovering biologically meaningful latent infection states. To accomplish this, scDEcrypter provides a built-in cross-validation procedure that evaluates a set of candidate lambda values using the held-out generation set created earlier.

Users may customize three aspects of lambda selection:

`lambda.cands`: A grid of candidate lambda values: this grid can span large orders of magnitude because the optimal penalty varies across datasets.

`max.iter`: The maximum number of EM iterations, controls how long the model is allowed to iterate for each lambda value.

`tol`: A convergence tolerance determines when the EM algorithm has stabilized.

All other required inputs, such as stabilized expression values, label vectors, and data splits, were already constructed in earlier steps.

```
lambdas <- c(1e2, 1e1, 1, 1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8) # wide range

# Running reduced range for the tutorial:
cv_lambda <- cross_validate_lambda(Data = seurat_train[top_genes.HVG,],
                                   lambda.cands = lambdas[1:2],
                                   max.iter = 2,
                                   tol = 1e-8,
                                   c_star = c_star, v_star = v_star,
                                   NCORES = 1)
```

```
## [Fold 1] Starting...
```

```
## Initializing...
## EM Iterating...
## 0.04841433
## 0.03472599
## Preparing outputs...
## Initializing...
## EM Iterating...
## 0.04841433
## 0.03472599
## Preparing outputs...
## [Fold 1] Fitted model, evaluating lambdas...
## [Fold 1] Complete
## [Fold 2] Starting...
## Initializing...
## EM Iterating...
## 0.06722278
## 0.05983655
## Preparing outputs...
## Initializing...
## EM Iterating...
## 0.06722278
## 0.05983655
## Preparing outputs...
## [Fold 2] Fitted model, evaluating lambdas...
## [Fold 2] Complete
## [Fold 3] Starting...
## Initializing...
## EM Iterating...
## 0.06671917
## 0.06557934
## Preparing outputs...
## Initializing...
## EM Iterating...
## 0.06671917
## 0.06557934
## Preparing outputs...
## [Fold 3] Fitted model, evaluating lambdas...
## [Fold 3] Complete
```

```
## [Fold 4] Starting...
## Initializing...
## EM Iterating...
## 0.05945427
## 0.05634428
## Preparing outputs...
## Initializing...
## EM Iterating...
## 0.05945427
## 0.05634428
## Preparing outputs...
## [Fold 4] Fitted model, evaluating lambdas...
## [Fold 4] Complete
## [Fold 5] Starting...
## Initializing...
## EM Iterating...
## 0.06639039
## 0.06344181
## Preparing outputs...
## Initializing...
## EM Iterating...
## 0.06639039
## 0.06344181
## Preparing outputs...
## [Fold 5] Fitted model, evaluating lambdas...
## [Fold 5] Complete
## Best lambda selected:  10
best_lambda <- cv_lambda$best_lambda
```

Fit the scDEcrypter multiway mixture model

After selecting the optimal penalty parameter, we fit the full scDEcrypter model using the generation set. Recall that the generation set was set aside specifically for model training to avoid using the same data for both parameter estimation and downstream inference. In this step, scDEcrypter estimates all components of the penalized multiway mixture model described in the manuscript. These fitted parameters are essential for:

1. Inferring probabilistic cell state weights for cells whose viral status is unknown (NA in V.obs)
2. Performing differential expression analysis based on the inferred infection groups

The results_generation object stores:

(1). conditional probabilities for latent infection states,

(2). estimated mixture proportions for each (infection × partition) combination,

(3). penalized mean and variance parameters,

(4). convergence diagnostics and log-likelihood trajectories.

These estimates form the foundation for the next steps.

scDEcrypter internally requires that the partitioning variable (e.g., cell type) be coded numerically. However, for interpretation, visualization, and reporting of results, it is far more informative to display human-readable category names instead of integers. We strongly advise supplying the labels that align with the pre-labeling scheme for C.preLabel and V.preLabel.

```
infection_labels <- c("Infected", "Uninfected", "Bystander")
celltype_labels <- make.names(levels(factor(seu_sub$ctype))) #removes spaces in names

results_generation <- fit_scDEcrypter(Data = seurat_train[top_genes.HVG,],
                                      infectionLabels = infection_labels,
                                      partitionLabels = celltype_labels,
                                      max.iter = 2, tol = 1e-8,
                                      c_star = c_star, v_star = v_star,
                                      lambda.vec = best_lambda)
```

```
## Initializing...
## EM Iterating...
## 0.06531182
## 0.06734417
## Preparing outputs...
```

Inferring test set cell weights

Exploring infection stateand/or additional partition variables weight

After estimating the multiway mixture model on the generation set, scDEcrypter can be used to infer probability weights for infection states for all cells by using their conditional probabilities across the latent viral-status classes. These conditional probabilities summarize how strongly each cell is supported as "infected," "uninfected," or "bystander", based on both its gene expression and the structure of the mixture model. These are obtained by running the getTestWeights function on the scDEcrypter outputs and the test dataset and stored as weights_test.

```
testData <- subset(seu_sub.processed, Index_Split == "Test")

results_generation <- getTestWeights(results_generation, testData)

names(results_generation)
```

```
## [1] "M_generation"       "sigma2_generation"  "probs_generation"
## [4] "weights_generation" "weights_test"
```

Applying a conditional probability threshold explores how cells are weighted across classes via the thresholdScoring function. If a cell has conditional probability cutoffs for one class, it is labeled as belonging to that class. A higher threshold (e.g., 0.9) leads to more conservative labeling—only the cells with the strongest evidence will be assigned.

Similarly, the above applies to cell-type assignment if cell-types were partially pre-labeled. Users may specify a threshold or set the option cutoffPartition to "max" indicating a cell should be assigned to the most likely cell-type, that is, the one it has the maximum estimated probability.

thresholdScoring outputs the seurat object with cell-state assignment columns added to the meta-data named: pred_InfectionState and pred_PartitionState. Cells that do not have a weight above the threshold or are exactly tied, receive a state of "NotAssigned".

```
testData_preds <- thresholdScoring(results_generation, testData,
                                   cutoffInfection = .9,
                                   cutoffPartition = "max", # not used here
                                   independentThres = TRUE)
```

```
## Predicting on test set...
```

```
table(testData_preds$pred_InfectionState, testData_preds$Sample)
```

```
##
##                DPI3 MOCK
##   Bystander     186    0
##   Infected      204    0
##   NotAssigned    73    0
##   Uninfected     97  440
```

Differential Expression Analysis

Select highly variable genes in the test set

Although HVGs were already selected for model training on the generation set, DE analysis may benefit from considering a broader pool of variable genes across infection states. The following procedure identifies genes for differential expression testing on the test dataset. We recommend this being large, at least 3,000-5,000.

```
top_genes.HVG_test <- get_top_genes(seurat_obj = subset(seurat_train,
    Sample == "DPI3"), partition_colname = "C.preLabel",
    n_genes_per_group = 500)  # to save computation time for vignette
length(top_genes.HVG_test)
```

```
## [1] 1334
```

Likelihood-ratio test for differential expression

scDEcrypter performs a cell-type–specific likelihood-ratio test (LRT) to evaluate whether gene expression differs between the two infection states (e.g., infected vs. bystander) within each cell type. The LRT compares:

1. Alternative model: gene expression depends on infection status,
2. Null model: gene expression does not depend on infection status.

To control the false-discovery rate across thousands of genes, scDEcrypter applies the Benjamini–Hochberg (BH) FDR procedure. In addition to significance testing, scDEcrypter computes the estimated log fold-change in expression between infection states within each cell type.

```
testData_de <- differentialResults(mod_results = results_generation,
    testData = testData, testingGenes = top_genes.HVG_test,
    compGroups = c("Infected", "Bystander"))
```

```
## Calculating under the alternative ...
```

```
## Warning in approx_complete_data_loglik(Y_test, M.alt, W.test, sigma2.alt, :
## Number of unique v_obs labels does not match that in the mean matrix.
```

```
## Calculating under the null ...
```

```
## Warning in approx_complete_data_loglik(Y_test, M.null, W.test, sigma2.null, :
## Number of unique v_obs labels does not match that in the mean matrix.
```

```
## Calculating test statistics and p-value ...
```

```
names(testData_de)
```

```
## [1] "ll.alternative" "ll.null"        "lrt.stat"       "pval"
## [5] "adj.pval"       "logfc"
```

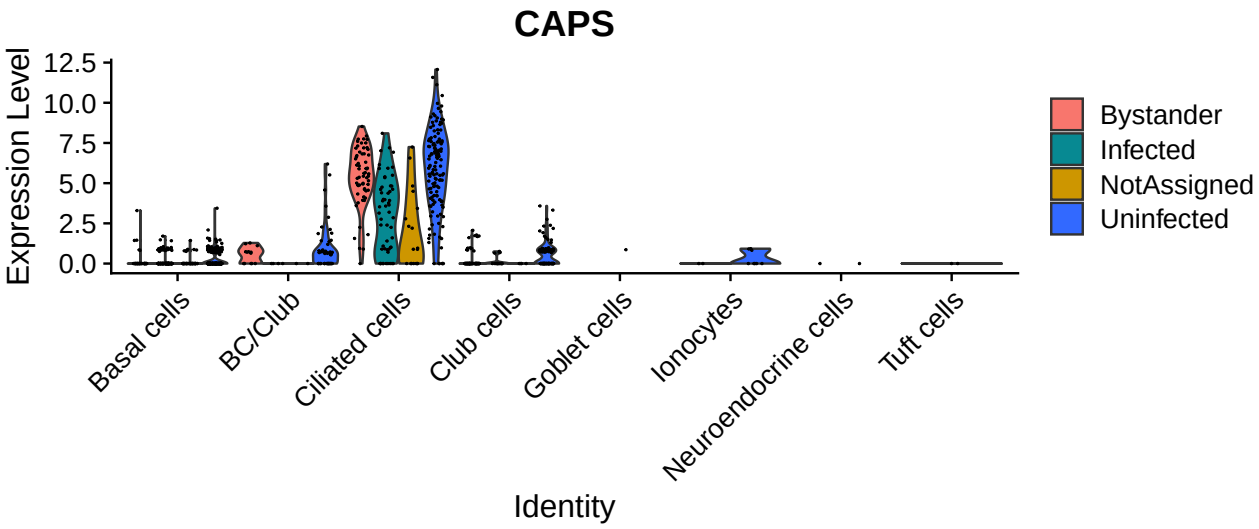
```
head(testData_de$logfc[order(testData_de$logfc$logFC_Ciliated.cells_Infected_vs_Bystander),
    1:3])
```

```
##        logFC_Basal.cells_Infected_vs_Bystander
## CAPS                                0.01661500
## TUBB4B                             -0.57354675
## CD24                               -0.87386501
## PRDX5                              -0.30403685
## TUBA1A                             -0.08662916
## DYNLL1                             -0.51813562
##        logFC_BC.Club_Infected_vs_Bystander
## CAPS                           -0.4863245
## TUBB4B                         -0.4552761
## CD24                           -0.4497149
## PRDX5                          -0.2960976
## TUBA1A                          0.1003466
## DYNLL1                         -0.6175406
##        logFC_Ciliated.cells_Infected_vs_Bystander
## CAPS                                    -2.459949
## TUBB4B                                  -2.295713
## CD24                                    -2.101314
## PRDX5                                   -2.097587
## TUBA1A                                  -2.041063
## DYNLL1                                  -2.027110
```

```
Seurat::VlnPlot(testData_preds, features = "CAPS",
    group.by = "ctype", split.by = "pred_InfectionState")
```

```
## The default behaviour of split.by has changed.
## Separate violin plots are now plotted side-by-side.
## To restore the old behaviour of a single split violin,
## set split.plot = TRUE.
##
## This message will be shown once per session.
```

```
## Warning: Groups with fewer than two datapoints have been dropped.
## i Set `drop = FALSE` to consider such groups for position adjustment purposes.
## Groups with fewer than two datapoints have been dropped.
## i Set `drop = FALSE` to consider such groups for position adjustment purposes.
## Groups with fewer than two datapoints have been dropped.
## i Set `drop = FALSE` to consider such groups for position adjustment purposes.
```



Session Info

```
sessionInfo()
```

```
## R version 4.4.1 (2024-06-14)
## Platform: x86_64-apple-darwin20
## Running under: macOS Ventura 13.7.8
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.4-x86_64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.4-x86_64/Resources/lib/libRlapack.dylib;  LAPACK v
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] scDEcrypter_0.3.0 devtools_2.4.5    usethis_3.0.0
##
## loaded via a namespace (and not attached):
##   [1] RcppAnnoy_0.0.22       splines_4.4.1          later_1.4.1
##   [4] tibble_3.2.1           polyclip_1.10-7        hardhat_1.4.0
##   [7] pROC_1.18.5            rpart_4.1.23           fastDummies_1.7.5
##  [10] lifecycle_1.0.4        pbmcapply_1.5.1        globals_0.16.3
##  [13] lattice_0.22-6         MASS_7.3-65            magrittr_2.0.3
##  [16] plotly_4.10.4          rmarkdown_2.29         yaml_2.3.10
##  [19] remotes_2.5.0          httpuv_1.6.15          Seurat_5.4.0
##  [22] sctransform_0.4.1      spam_2.11-1            sp_2.2-0
##  [25] sessioninfo_1.2.2      pkgbuild_1.4.4         spatstat.sparse_3.1-0
##  [28] reticulate_1.41.0      cowplot_1.1.3          pbapply_1.7-2
##  [31] RColorBrewer_1.1-3     lubridate_1.9.3        abind_1.4-8
##  [34] pkgload_1.4.0          zlibbioc_1.50.0        Rtsne_0.17
##  [37] purrr_1.0.4            BiocGenerics_0.56.0    nnet_7.3-19
```

```
##  [40] ipred_0.9-15           lava_1.8.0             IRanges_2.38.1
##  [43] S4Vectors_0.48.0       ggrepel_0.9.6          irlba_2.3.5.1
##  [46] listenv_0.9.1          spatstat.utils_3.1-2   goftest_1.2-3
##  [49] RSpectra_0.16-2        spatstat.random_3.3-2  fitdistrplus_1.2-2
##  [52] parallelly_1.42.0      codetools_0.2-20       DelayedArray_0.30.1
##  [55] tidyselect_1.2.1       farver_2.1.2           matrixStats_1.5.0
##  [58] stats4_4.4.1           spatstat.explore_3.3-4 jsonlite_1.9.0
##  [61] caret_6.0-94           ellipsis_0.3.2         progressr_0.15.1
##  [64] ggridges_0.5.6         survival_3.7-0         iterators_1.0.14
##  [67] foreach_1.5.2          tools_4.4.1            ica_1.0-3
##  [70] Rcpp_1.0.14            glue_1.8.0             prodlim_2024.06.25
##  [73] gridExtra_2.3          SparseArray_1.4.8      xfun_0.51
##  [76] MatrixGenerics_1.16.0  dplyr_1.1.4            withr_3.0.2
##  [79] formatR_1.14           fastmap_1.2.0          digest_0.6.37
##  [82] timechange_0.3.0       R6_2.6.1               mime_0.12
##  [85] scattermore_1.2        transformGamPoi_1.10.0 tensor_1.5
##  [88] dichromat_2.0-0.1      spatstat.data_3.1-4    tidyr_1.3.1
##  [91] generics_0.1.3         data.table_1.17.4      recipes_1.1.0
##  [94] class_7.3-22           httr_1.4.7             htmlwidgets_1.6.4
##  [97] S4Arrays_1.4.1         uwot_0.2.3             ModelMetrics_1.2.2.2
## [100] pkgconfig_2.0.3        gtable_0.3.6           timeDate_4041.110
## [103] lmtest_0.9-40          XVector_0.44.0         htmltools_0.5.8.1
## [106] profvis_0.4.0          dotCall64_1.2          SeuratObject_5.0.2
## [109] scales_1.4.0           png_0.1-8              gower_1.0.1
## [112] spatstat.univar_3.1-1  knitr_1.49             rstudioapi_0.17.1
## [115] reshape2_1.4.4         nlme_3.1-166           cachem_1.1.0
## [118] zoo_1.8-13             stringr_1.5.1          KernSmooth_2.23-24
## [121] vipor_0.4.7            parallel_4.4.1         miniUI_0.1.1.1
## [124] ggrastr_1.0.2          pillar_1.10.2          grid_4.4.1
## [127] vctrs_0.6.5            RANN_2.6.2             urlchecker_1.0.1
## [130] promises_1.3.2         xtable_1.8-4           cluster_2.1.6
## [133] beeswarm_0.4.0         evaluate_1.0.3         tinytex_0.56
## [136] cli_3.6.5              compiler_4.4.1         rlang_1.1.6
## [139] crayon_1.5.3           future.apply_1.11.3    labeling_0.4.3
## [142] ggbeeswarm_0.7.2       plyr_1.8.9             fs_1.6.5
## [145] stringi_1.8.4          viridisLite_0.4.2      deldir_2.0-4
## [148] lazyeval_0.2.2         spatstat.geom_3.3-5    Matrix_1.7-1
## [151] RcppHNSW_0.6.0         patchwork_1.3.0        future_1.34.0
## [154] ggplot2_3.5.2          shiny_1.10.0           ROCR_1.0-11
## [157] igraph_2.1.4           memoise_2.0.1
```