scDEcrypter: Detecting latent viral states in scRNA-seq

Luer Zhong

2025-11-20

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 other partitioning variables such as cell type. The method uses a penalized multiway 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. If you use scDEcrypter in published research, please cite:

Run scDEcrypter

Install the package

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

```r
library(devtools)
devtools::install_github("https://github.com/LZHONG25/scDEcrypter")
library(scDEcrypter)
```

Required inputs

Load the Seurat object. We analyzed the COVID-19 nasal wash dataset described in
Gao, Kevin M., *et al.* (2021). *Human nasal wash RNA-Seq reveals distinct cell-specific innate immune responses in influenza versus SARS-CoV-2.* **JCI Insight**, 6(22): e152288.

```r
seed <- sample(63321232,1)
set.seed(seed)
load(".../covid_full_raw_seu.Rdata")
# the seurat object covid_full_raw_seu.Rdata is saved as seurat_obj
```

Define the partitioning variable

scDEcrypter requires one additional cell-level label, which we refer to as the 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 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 believes is relevant.

In the example shown here, we use broad cell-type categories as the partitioning variable. Fine-grained annotations from the Seurat object are collapsed into six coarse groups (T cells, macrophages, B cells, epithelial cells, dendritic cells, and neutrophils). Users may adapt or replace this mapping entirely depending on the structure of their own dataset. Make sure that the partitioning variable is numerically coded.

```r
seurat_obj$CellType_Label <- ifelse(seurat_obj$CellType_Fine %in%
    c("CD4_T", "CD8_T", "prolifT"), 1, ifelse(seurat_obj$CellType_Fine %in%
    c("alveol-mac", "IFNexp-mac", "M1-mac",
        "M1-mac-exp", "M2-mac"), 2, ifelse(seurat_obj$CellType_Fine %in%
    c("memoryB1", "naiveB", "transB", "plasma",
        "ABC"), 3, ifelse(seurat_obj$CellType_Fine %in%
    c("basal", "ciliated", "goblet+club",
        "hillock"), 4, ifelse(seurat_obj$CellType_Fine %in%
    c("pDC"), 5, ifelse(seurat_obj$CellType_Fine %in%
    c("G5a_naive", "G5b", "G5c_aged", "G5c_naive"),
    6, NA))))))
```

Define the viral status variable

scDEcrypter requires a cell-level viral status variable (V.obs) that indicates whether each cell is infected or uninfected. In real scRNA-seq viral infection datasets, this information is rarely fully observed:

(1). a small subset of cells have strong viral read evidence and can be confidently labeled as infected,

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

(3). some cells fall somewhere in between, where the absence of viral reads does not guarantee that the cell is truly uninfected (e.g., low viral load or bystander cells).

scDEcrypter is explicitly designed to model this partially latent structure, as described in the paper. These confidently labeled cells serve as anchors for the mixture model, while the remaining cells are treated as unknown (NA) and their infection status is inferred probabilistically during model fitting. To provide this information to scDEcrypter, users should:

(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:

```r
C.obs <- seurat_obj$CellType_Label

V.obs <- rep(NA, ncol(seurat_obj))
to_label_I <- which(seurat_obj$Infection_byViralCounts == "Infected")
to_label_U <- which(seurat_obj$Group == "Healthy")
V.obs[to_label_I] <- 1
V.obs[to_label_U] <- 2
```

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, bystander), and a partitioning variable class (e.g., cell type, biological group, or any other cell-level label).

To fit this model, users must 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. As described in the paper, for the COVID-19 nasal wash dataset, we observed: Infected cells (with viral reads), Uninfected cells (from healthy donors), and Bystander cells (from infected individuals, but lacking viral reads). Because distinguishing bystander cells was biologically meaningful, we set the total number of viral status categories as 3. 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. For this dataset, after grouping fine-grained cell annotations into broader categories, we determined there were 6 cell type categories.

```r
V.star <- 3
C.star <- 6
```

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 multi-way 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:

1. Splits cells into generation/test sets (stratified by known infection and partitioning labels when possible);
2. Applies normalization + variance stabilization independently to each split;
3. Returns variance-stabilized matrices for both sets;
4. Returns the indexing and label information required for model training.

Variance stabilization is an essential part of the scDEcrypter workflow: the underlying mixture model assumes approximately Gaussian stabilized expression values. Users can specify their preferred method via vs_method, In this vignette, we use "shifted_log".

```r
prep <- preprocess_scDEcrypter(seurat_obj, C.obs, V.obs, seed, vs_method = "shifted_log")

Y_generation_stbl <- prep$Y_generation
Y_test_stbl <- prep$Y_test
splitted_data <- prep$splitted
```

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 test set and preserves the integrity of the data-splitting framework described in the manuscript. For the COVID example, we used a cell-type–aware HVG selection procedure - within each cell type, among COVID-positive samples only, and selected the most variable genes per cell type, then aggregated them. Users may replace the above with any 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.

```r
vargroup <- na.omit(seurat_obj$CellType_Label[splitted_data$generation_idx])
covgroup <- names(which(seurat_obj$Group[splitted_data$generation_idx] == "COVID-19"))

top_genesA <- sapply(unique(vargroup), function(x)
  apply(Y_generation_stbl[intersect(covgroup, names(which(vargroup==x))),], 2, var))

topidx <- apply(top_genesA, 2, function(x) order(x, decreasing = T)[1:500])
topidx_g <- apply(topidx, 2, function(x) rownames(top_genesA)[x])
top_genes.HVG <- unique(as.vector(apply(topidx_g, 2, function(x) x[1:50])))
```

Select the best 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:

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

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

A convergence tolerance (tol): 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.

```r
lambda.vec <- c(1e2, 1, 0, 1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8)
cv_lambda <- cross_validate_lambda(Y_generation_stbl[,top_genes.HVG],
                                   splitted_data$C.obs_generation,
                                   splitted_data$V.obs_generation,
                                   lambda.vec, max.iter = 200,
                                   tol = 1e-8, C.star, V.star, seed,
                                   NCORES = 5)
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. Predicting latent infection states 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). posterior probabilities for latent infection states,

(2). estimated mixture proportions for each (infection $\times$ partition) combination,

(3). penalized mean and variance parameters,

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

These estimates form the foundation for the next steps: predicting infection status for all cells and conducting cell-type–specific differential expression testing.

```r
results_generation <- MultiwayMixture(Y = Y_generation_stbl[,top_genes.HVG],
                                      C.obs = splitted_data$C.obs_generation,
                                      V.obs = splitted_data$V.obs_generation,
                                      max.iter = 200, tol = 1e-8,
                                      C.star, V.star, seed,
                                      lambda.vec=dim(splitted_data$Y_generation)[1]
                                      *best_lambda)
```

```r
cutoffs <- .9
cell_thresh <- .75
```

```r
M.1 <- results_generation$M_list[[1]]
sigma2.1 <- results_generation$sigma2_list[[1]]
probs.1 <- results_generation$probs_list[[1]]
W.1 <- results_generation$weights_list[[1]]

W.2.test <- E.step1(Y_test_stbl[, top_genes.HVG], splitted_data$C.obs_test,
    splitted_data$V.obs_test, M.1, probs.1, sigma2.1)

W_max_byC <- apply(W.2.test, c(1, 2), max, na.rm = TRUE)
CellType_pred <- max.col(W_max_byC, ties.method = "first")
CellType_pred[which(apply(W_max_byC, 1, max) < cell_thresh)] <- 0

temp1 <- as.data.frame(W.2.test[, , 1, drop = F])  # infected
temp2 <- as.data.frame(W.2.test[, , 2, drop = F])  # uninfected
temp3 <- as.data.frame(W.2.test[, , 3, drop = F])  # bystander

temp1$Prediction_Infected <- (ifelse(temp1$V1 >= cutoffs |
    temp1$V2 >= cutoffs | temp1$V3 >= cutoffs | temp1$V4 >=
    cutoffs | temp1$V5 >= cutoffs | temp1$V6 >= cutoffs,
    "Infected", "Unknown"))
```

```r
temp2$Prediction_Uninfected <- (ifelse(temp2$V1 >=
    cutoffs | temp2$V2 >= cutoffs | temp2$V3 >= cutoffs |
    temp2$V4 >= cutoffs | temp2$V5 >= cutoffs | temp2$V6 >=
    cutoffs, "Uninfected", "Unknown"))
temp3$Prediction_Bystander <- (ifelse(temp3$V1 >= cutoffs |
    temp3$V2 >= cutoffs | temp3$V3 >= cutoffs | temp3$V4 >=
    cutoffs | temp3$V5 >= cutoffs | temp3$V6 >= cutoffs,
    "Bystander", "Unknown"))
temp1$Full_Prediction <- "Unknown"
temp1$Full_Prediction[which(temp1$Prediction_Infected ==
    "Infected")] <- "Infected"
temp1$Full_Prediction[which(temp2$Prediction_Uninfected ==
    "Uninfected")] <- "Uninfected"
temp1$Full_Prediction[which(temp3$Prediction_Bystander ==
    "Bystander")] <- "Bystander"
```