Batch-effect correction in single-cell RNA sequencing data using JIVE

Joseph Hastings

2023-04-20

Overview

- Introduction
  - Single-cell RNA Sequencing
  - Batch-Effect Correction Methods
  - JIVE
- Methods
  - scRNA-seq Datasets
  - JIVE Algorithm Improvements
  - Batch Correction Evaluation Metrics
- Results
  - JIVE Runtime Improvements
  - Simulated Data
  - Assessments Using Real Data
- Discussion

Introduction

Single-cell RNA Sequencing

RNA Sequencing Background

- Technique used to study the expression and regulation of genes
- Isolates and sequences RNA molecules
- Comprehensive view of the transcriptome
- Many applications:
  - Gene expression analysis
  - Differential gene expression analysis
  - Functional annotation
  - Biomarker discovery

Single-cell RNA Sequencing

Bulk RNA Sequencing versus Single-cell RNA Sequencing

- Resolution
  - Entire population of cells vs. individual cells
- Cell number
  - Large number of cells (tens of thousands to millions) vs. fewer number of cells required (hundreds to thousands)
- Coverage
  - Measures average gene expression across all cells in a sample vs. measures expression of genes in each individual cell
- Cost
  - Bulk RNA-seq less expensive per cell, cost per sample higher compared to scRNA-seq
- Data analysis
  - scRNA-seq data requires more specialized tools

Single-cell RNA Sequencing

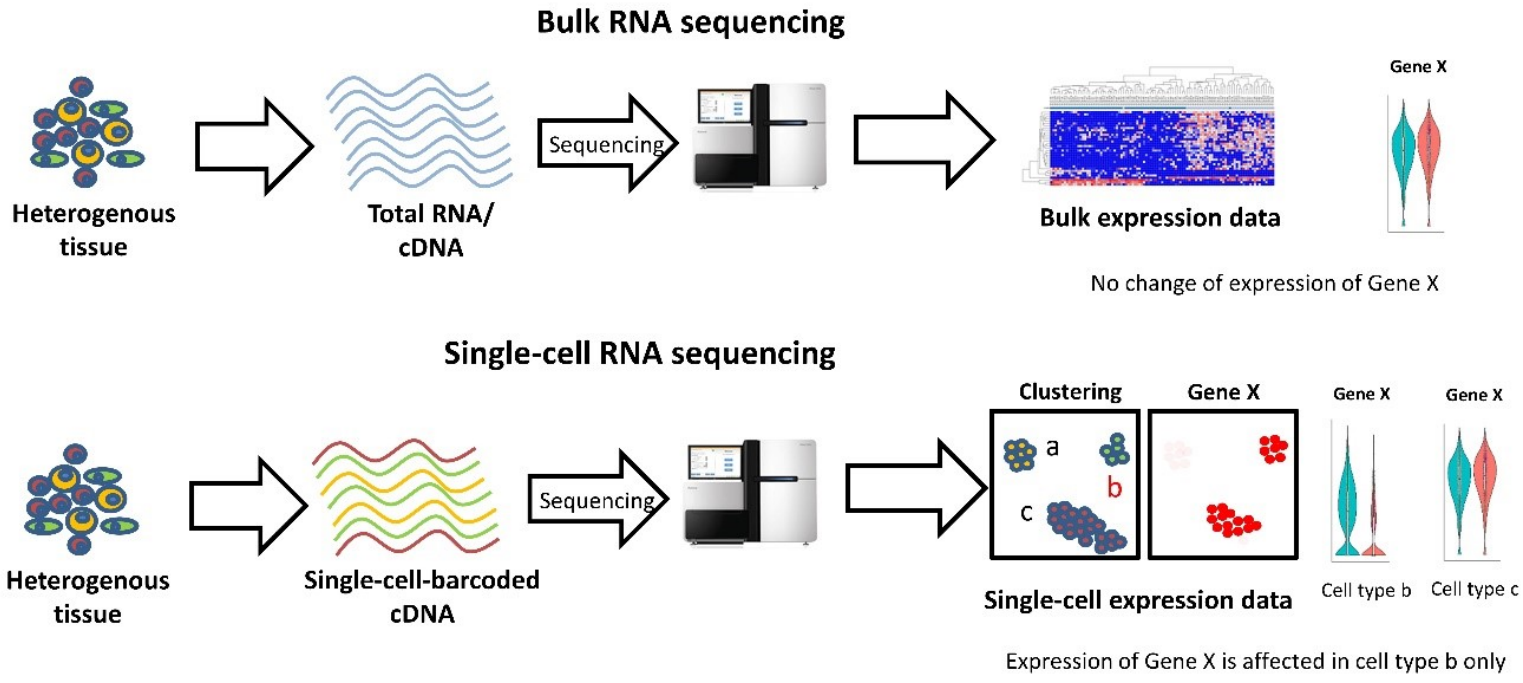

Figure 1: Bulk RNA-seq vs. scRNA-seq

Single-cell RNA Sequencing

Data

- scRNA-seq data consists of a count matrix
- Rows represent genes, columns represent cells
- Library size

Preprocessing

- Normalization
  - Counts per million (CPM)
  - Upper quartile (UQ)
  - Trimmed mean of M values (TMM)

Single-cell RNA Sequencing

Dataset Integration

- Cell counts can be obtained from different batches
  - Batch effects
- Many methods developed to remove these batch effects

Batch-Effect Correction Methods

Seurat v3

- Software package developed by the Satija lab
- Graph based approach to dataset integration
- Maps cells in each dataset to their closest anchors
- Aligned datasets are combined into a single integrated dataset

Batch-Effect Correction Methods

Seurat v3

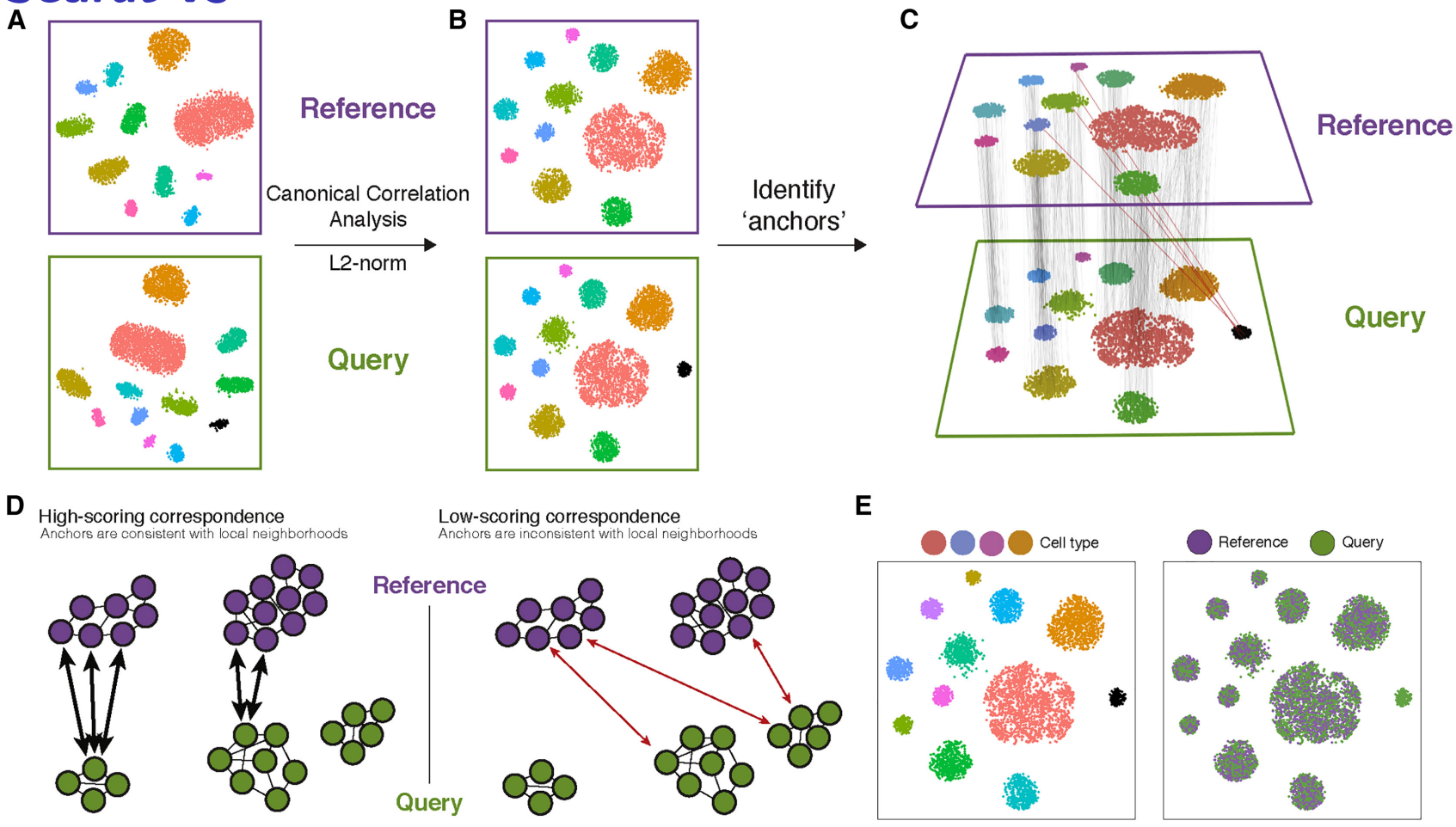

Figure 2: Seurat v3 Process

Batch-Effect Correction Methods

Harmony

- Utilizes low-dimensional representation of datasets
- Iterates between two algorithms until convergence:
  - Clustering which preserves diversity
  - Batch correction

Batch-Effect Correction Methods

Harmony

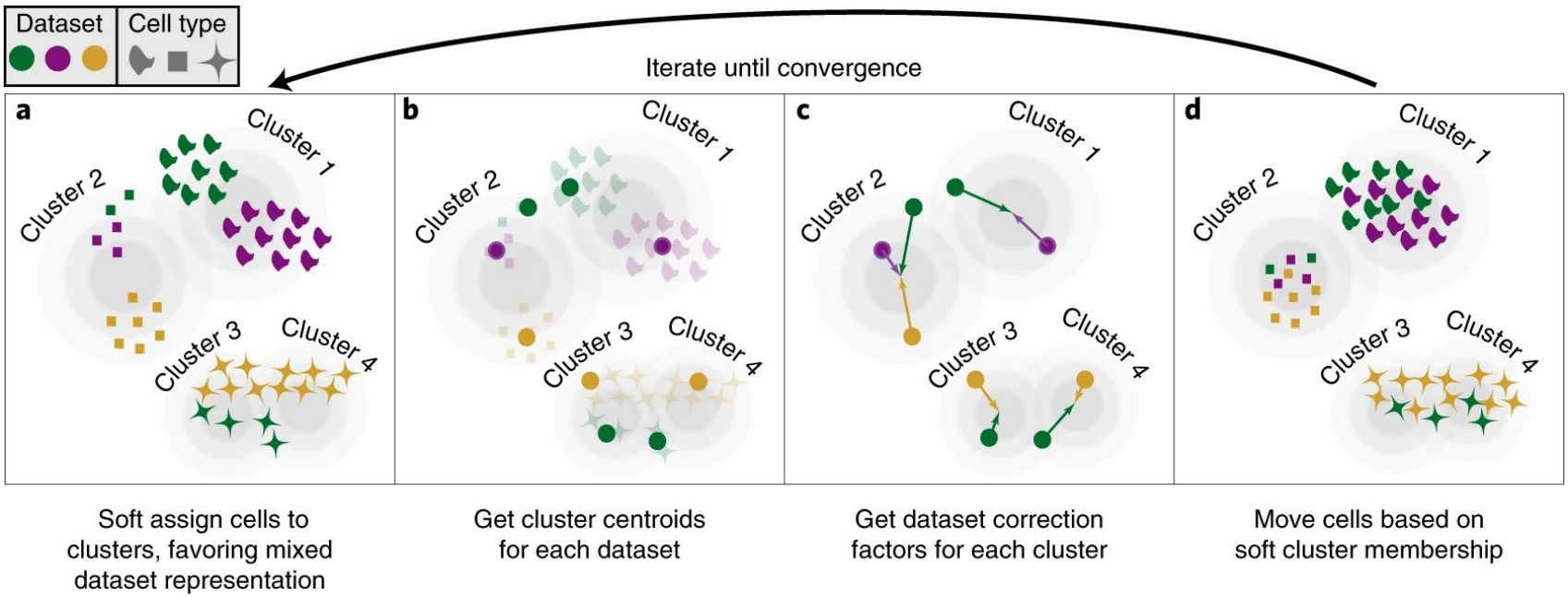

Figure 3: Harmony Integration Process

JIVE

Joint and individual variance explained (JIVE)

- Integrates multi-omics data by a decomposition method
  - Low-rank approximation for joint data variation
  - Low-rank approximations for individual dataset variations
  - Residual noise
- Simultaneous integration and dimension reduction
- Can enforce orthogonality between joint and individual structures
- Originally developed for bulk sequencing data

JIVE

Model

- Let $X_1, X_2, \ldots, X_k$ with $k \geq 2$ be matrices
- X_i has dimension $p_i \times n$
  - n columns corresponding to a common set of n objects
  - p_i rows corresponding to variables in a given measurement technology
  - $\sum p_i = p$
- Unified JIVE model:

$$\begin{aligned} X_1 &= J_1 + A_1 + \epsilon_1 \\ X_2 &= J_2 + A_2 + \epsilon_2 \\ &\vdots \\ X_k &= J_k + A_k + \epsilon_k \end{aligned} \tag{1}$$

JIVE

Model

- Joint structure is represented by J, a single $p \times n$ matrix of rank $r < \text{rank}(X)$

$$J = \begin{bmatrix} J_1 \\ \vdots \\ J_k \end{bmatrix}_{p \times n} \tag{2}$$

- J_i is the submatrix of J associated with X_i
- Individual structures for each X_i is represented by A_i, a $p_i \times n$ matrix of rank $r_i < \text{rank}(X_i)$
- ϵ_i are independent error matrices with $\mathbb{E}(\epsilon_i) = \mathbf{0}_{p_i \times n}$

JIVE

Relation to Principal Component Analysis

- Given a dataset $X_{p\times n}$ with variance-covariance matrix $\Sigma_{p\times p}$, perform an eigendecomposition on Σ:

$$\Sigma_{p\times p} = U_{p\times p} D_{p\times p} U_{p\times p}^T \tag{3}$$

- U denotes the p eigenvectors (known as loadings)
- D is the diagonal matrix containing the p eigenvalues of Σ

JIVE

Relation to Principal Component Analysis

- Can calculate a rank r approximation to X in the following way:

$$X_{p\times n} \approx U_{p\times r} S_{r\times n} \tag{4}$$

- U is the first r columns from the loading matrix
- S is the first r rows from the scores matrix

JIVE

Relation to Principal Component Analysis

- JIVE model can be written in a similar manner:

$$\begin{aligned} X_1 &\approx U_1 S + W_1 S_1 + R_1 \\ X_2 &\approx U_2 S + W_2 S_2 + R_2 \\ &\vdots \\ X_k &\approx U_k S + W_k S_k + R_k \end{aligned} \tag{5}$$

- U_i are the loadings of the joint structure corresponding to rows of X_i
- S is a common score matrix for the joint structure
- W_i is a $p_i \times r_i$ loading matrix, S_i is an $r_i \times n$ score matrix

JIVE

Estimation

- Estimates joint and individual structures by minimizing the sum of squared error:

$$R = \begin{bmatrix} R_1 \\ \vdots \\ R_k \end{bmatrix} = \begin{bmatrix} X_1 - J_1 - A_1 \\ \vdots \\ X_k - J_k - A_k \end{bmatrix} \tag{6}$$

- Given an estimate for J, find $A_1, \ldots, A_k$ to minimize $||R||$
- Given an estimate for $A_1, \ldots, A_k$, find J to minimize $||R||$
- Repeat until convergence

JIVE

Viable Batch-Effect Correction Method?

- JIVE estimates the joint and individual structures and performs dimension reduction simultaneously
  - Expect the joint structure to capture true biological signal
  - Expect the individual structures to capture technical effects
- Possible superior performance as less information is lost
  - Other methods "regress out" technical effects
  - In JIVE, effects not captured by joint structure will be in an individual structure, and vice versa
- We consider the common set of n objects to be genes measured in different batches
- We consider the p_i variables to be the different cells in batch i

Methods

scRNA-seq Datasets

- One simulated dataset and two real scRNA-seq datasets considered
- Two batches generated or selected from each with at least two different cell types
- Principal variance component analysis (PVCA) performed on each dataset
  - Utilizes principal components and variance components analysis
  - Fits a mixed linear model to partition the total observed variability into variability attributed to batch effects, cell type effects, and random error

scRNA-seq Datasets

Simulated Data

- `Splatter` R package used to simulate a scRNA-seq dataset
- Core of the model is a gamma-Poisson distribution to generate cell counts for a set number of genes
- More than 20 different parameters to modify
  - Library size, gene expression means, outlier probability, batch effects, and more

scRNA-seq Datasets

Simulated Data

- Generated two batches with 500 cells each
- Each batch contained three cell types in similar proportion
- Simple dataset to test the ability of JIVE

Table 1: Batch and Cell Type Frequency for Simulated Data

Batch Name	Group 1	Group 2	Group 3	Total
Batch1	312	136	52	500
Batch2	294	148	58	500

scRNA-seq Datasets

Simulated Data

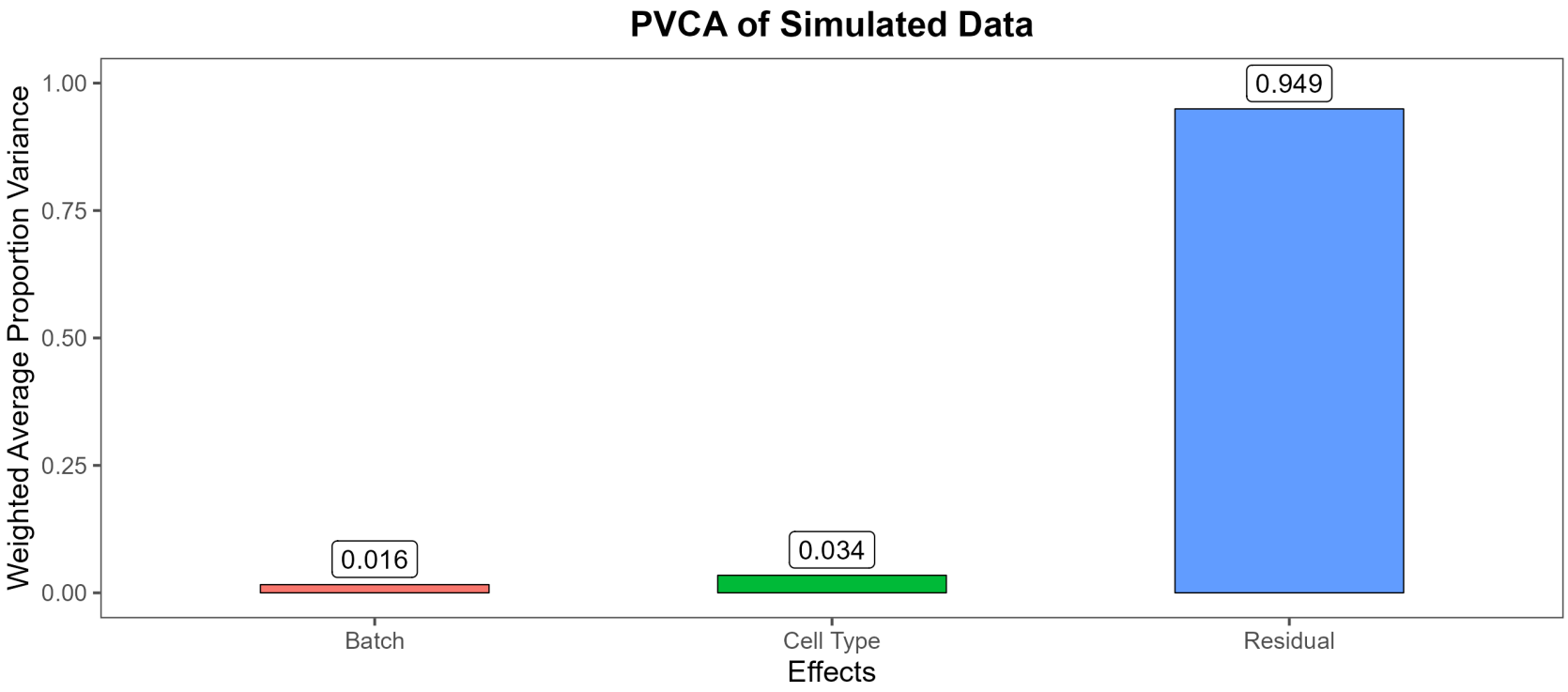

Figure 4: PVCA Breakdown for Simulated Data

scRNA-seq Datasets

Bacher T-Cell Data

- CD4+ T-Cell scRNA sequencing data obtained from six unexposed and fourteen COVID-19 positive patients
- Fifteen total batches and six different cell types provided
- Not a lot of distinction between batches and cell types, difficult to integrate

Table 2: Batch and Cell Type Frequency for Bacher T-Cell Data

Batch Name	Central Memory	Cycling	Cytotoxic / Th1	Tfh-like
14	230	1	76	250
15	239	4	143	425
	Transitional Memory	Type-1 IFN signature	Total	
14	190	7	754	
15	289	13	1113	

scRNA-seq Datasets

Bacher T-Cell Data

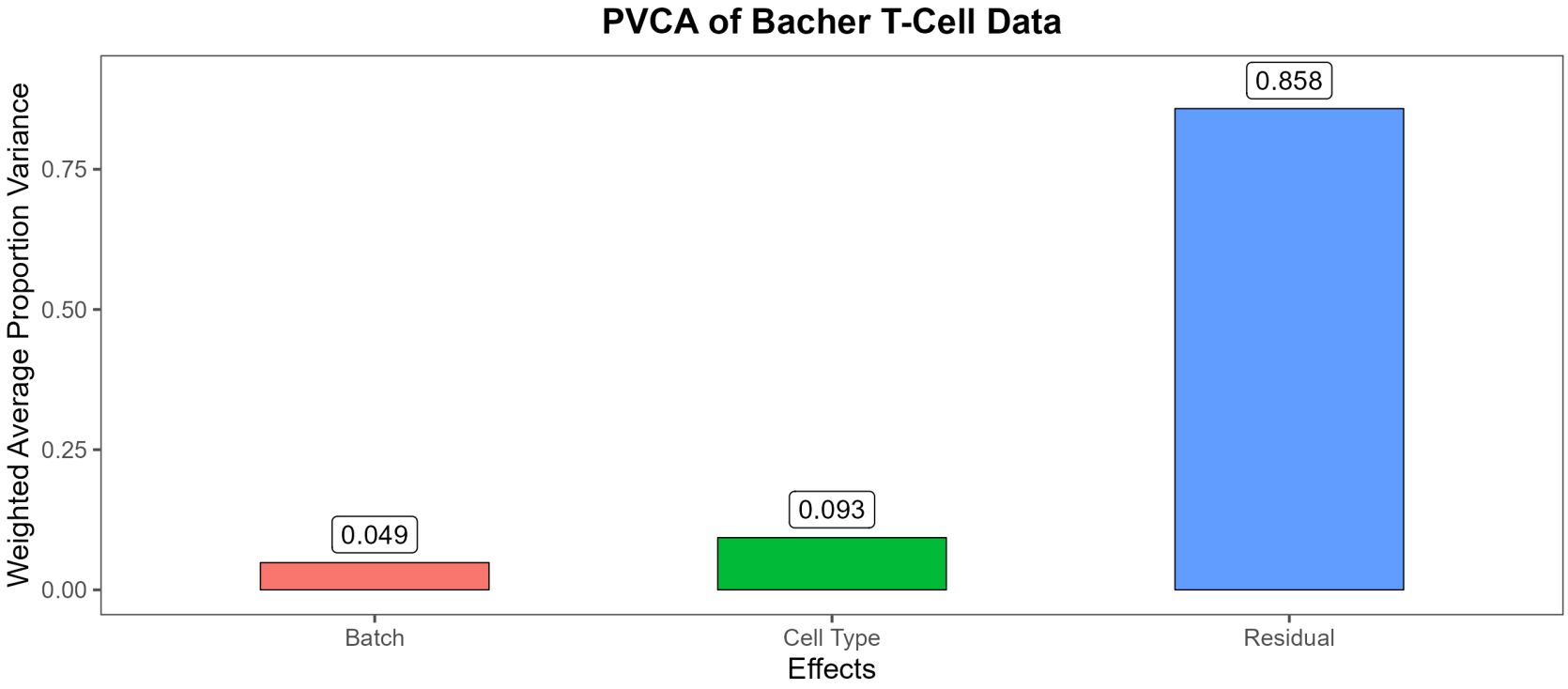

Figure 5: PVCA Breakdown for Bacher T-Cell Data

scRNA-seq Datasets

Zilionis Mouse Lung Data

- Analyzed tumor-infiltrating myeloid cells in mouse lung cancers
- Three total batches and seven cell types provided
- Distinct separation between batches and chosen cell types are not similar

Table 3: Batch and Cell Type Frequency for Zilionis Mouse Lung Data

Batch Name	B cells	T cells	Total
round1_20151128	641	579	1220
round2_20151217	879	434	1313

scRNA-seq Datasets

Zilionis Mouse Lung Data

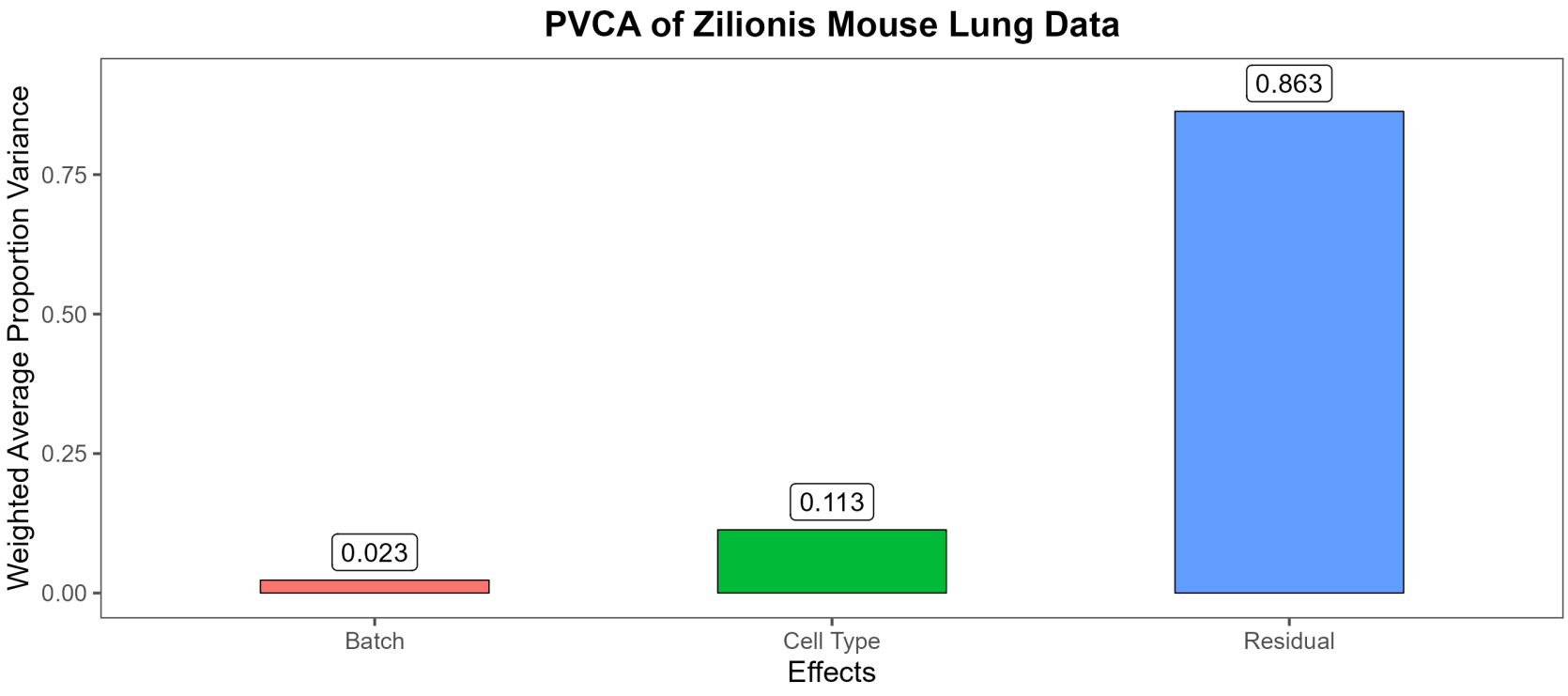

Figure 6: PVCA Breakdown for Zilionis Mouse Lung Data

JIVE Algorithm Improvements

- Original MATLAB code was implemented into the `R.JIVE` R package
  - Joint and individual structure estimation can take a considerable amount of time (12+ hours) depending on the size of the datasets
- We improved the speed of these algorithms in two ways:
  - Utilizing partial singular value decomposition (SVD) via the `RSpectra` R package
  - Converting frequently used matrix operations into precompiled C++ code using the `Rcpp` and `RcppEigen` R packages

JIVE Algorithm Improvements

Partial SVD

- Original `R.JIVE` code utilizes the SVD in multiple areas
  - Only the largest singular values/vectors are used
- Performing a full SVD takes a considerable amount of time and resources to compute
  - Majority of it is not even used

JIVE Algorithm Improvements

Partial SVD

- We switched to a partial SVD function in the `RSpectra` R package which calculates and returns the k largest singular values/vectors
- We performed a benchmark comparing this new function with the base R function
  - Generated a 1000×1000 matrix from a standard normal distribution
  - $k = 1$, 5, and 10 singular values/vectors were calculated for each function
  - Repeated this process 100 times for each function call, runtime was recorded

JIVE Algorithm Improvements

Matrix Multiplication

- Matrix multiplication is utilized constantly throughout the algorithm
- This was another area where runtime was getting bottle-necked in testing
- We tried different matrix multiplication functions
  - Armadillo C++ library via the `RcppArmadillo` R package
  - Eigen C++ library via the `RcppEigen` R package

JIVE Algorithm Improvements

Matrix Multiplication

- We switched to using a matrix multiplication function from the Eigen C++ library
- Allows the user to specify the number of CPU cores to use
- We performed a benchmark comparing the base %*% R operator to functions from the Armadillo and Eigen C++ libraries
  - Generated two 1000×1000 matrices
  - Performed matrix multiplication using each function
  - Repeated this process 100 times for each functoin call, runtime was recorded

JIVE Algorithm Improvements

Overall Improvements

- We compared the runtimes of the original `R.JIVE` functions to our updated versions
  - Two matrices A and B of size 200×1000 were generated from a common joint structure matrix, two unique individual structure matrices, and two residual error matrices generated from a standard normal distribution
  - Each decomposition was repeated 20 times, runtimes were recorded

JIVE Algorithm Improvements

Overall Improvements

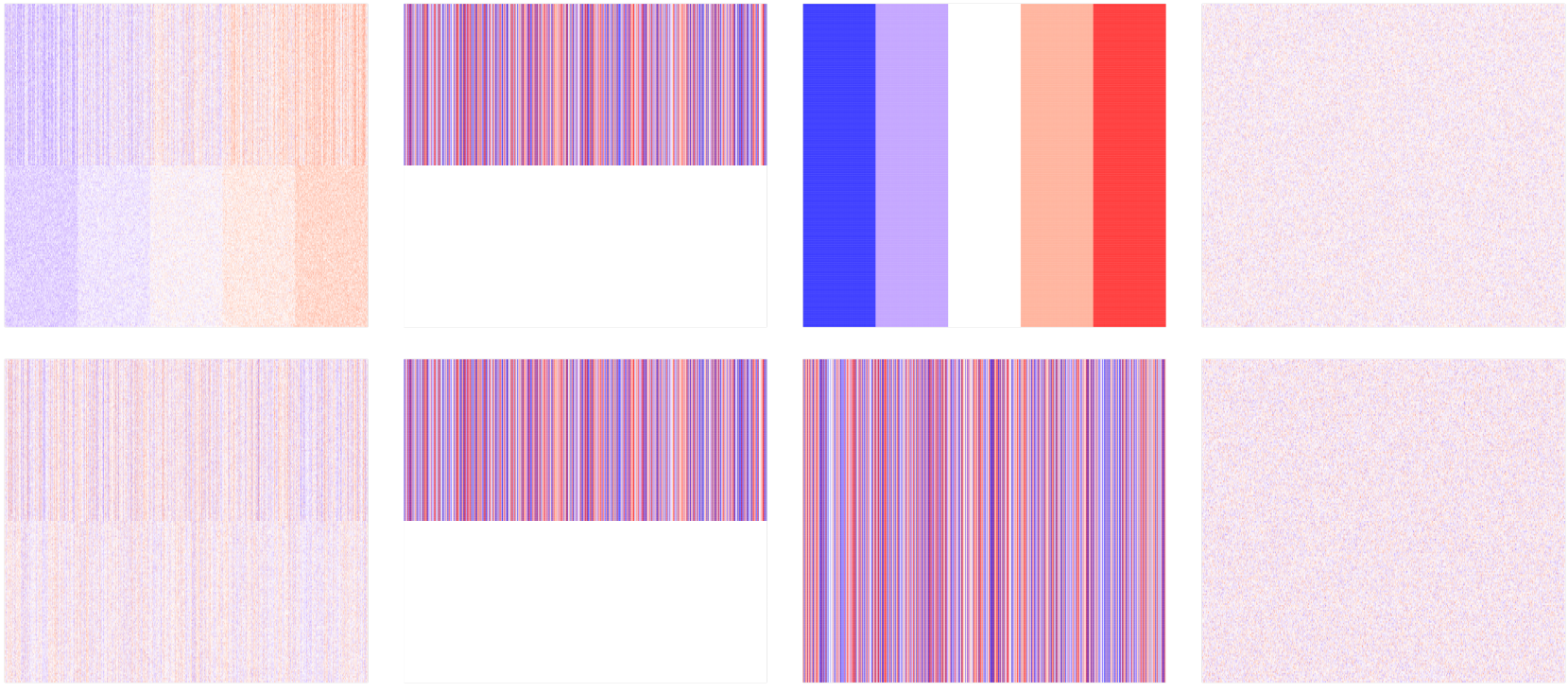

Figure 7: Heatmaps for simulation data used in JIVE benchmarks

Batch Correction Evaluation Metrics

We employed five tools/metrics to evaluate the performance of each batch correction method

- Qualitative Metrics:
  - t-distributed stochastic neighbor embedding (t-SNE) plots
  - uniform manifold approximation and project (UMAP) plots
- Quantitative Metrics
  - k-nearest neighbor batch effect test (kBET)
  - average silhouette width (ASW)
  - local inverse Simpson's index (LIS)

Batch Correction Evaluation Metrics

t-distributed stochastic neighbor embedding (t-SNE)

- t-SNE is a non-linear dimension reduction technique
- Assigns each data point from a high-dimensional dataset to a location in a two or three-dimensional map
- Aims to preserve as much of the local structure in original data as possible while also revealing global structure
  - High dimensional representation of data is used to create conditional probabilities of one point picking another as its neighbor
  - Similar conditional probability is calculated for a two to three-dimensional representation
  - Finds the low dimensional representation which best matches the these two probabilities for each point

Batch Correction Evaluation Metrics

t-distributed stochastic neighbor embedding (t-SNE)

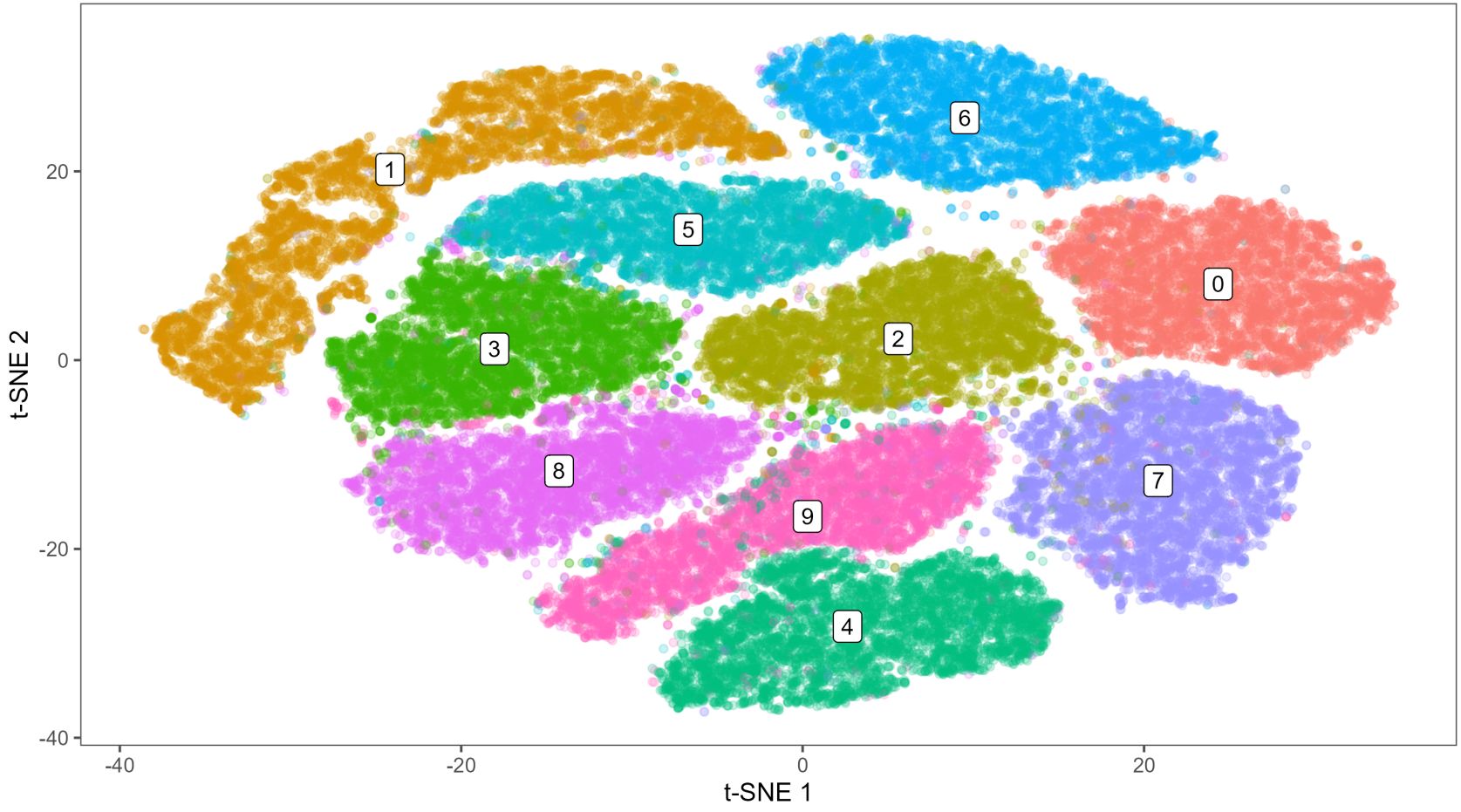

Figure 8: t-SNE Plot of MNIST Digit Data

Batch Correction Evaluation Metrics

Uniform manifold approximation and projection (UMAP)

- UMAP is another non-linear dimension reduction technique
- Based in manifold theory and topological analysis
- It's algorithm has two main phases: graph construction and graph layout
  - During graph construction, a weighted k-nearest neighbor graph is created and transformations are applied to the graph's edges
  - During graph layout, an objective function is defined that preserves important characteristics in the k-nearest neighbor graph
  - The final UMAP representation is the one which minimizes this function

Batch Correction Evaluation Metrics

Uniform manifold approximation and projection (UMAP)

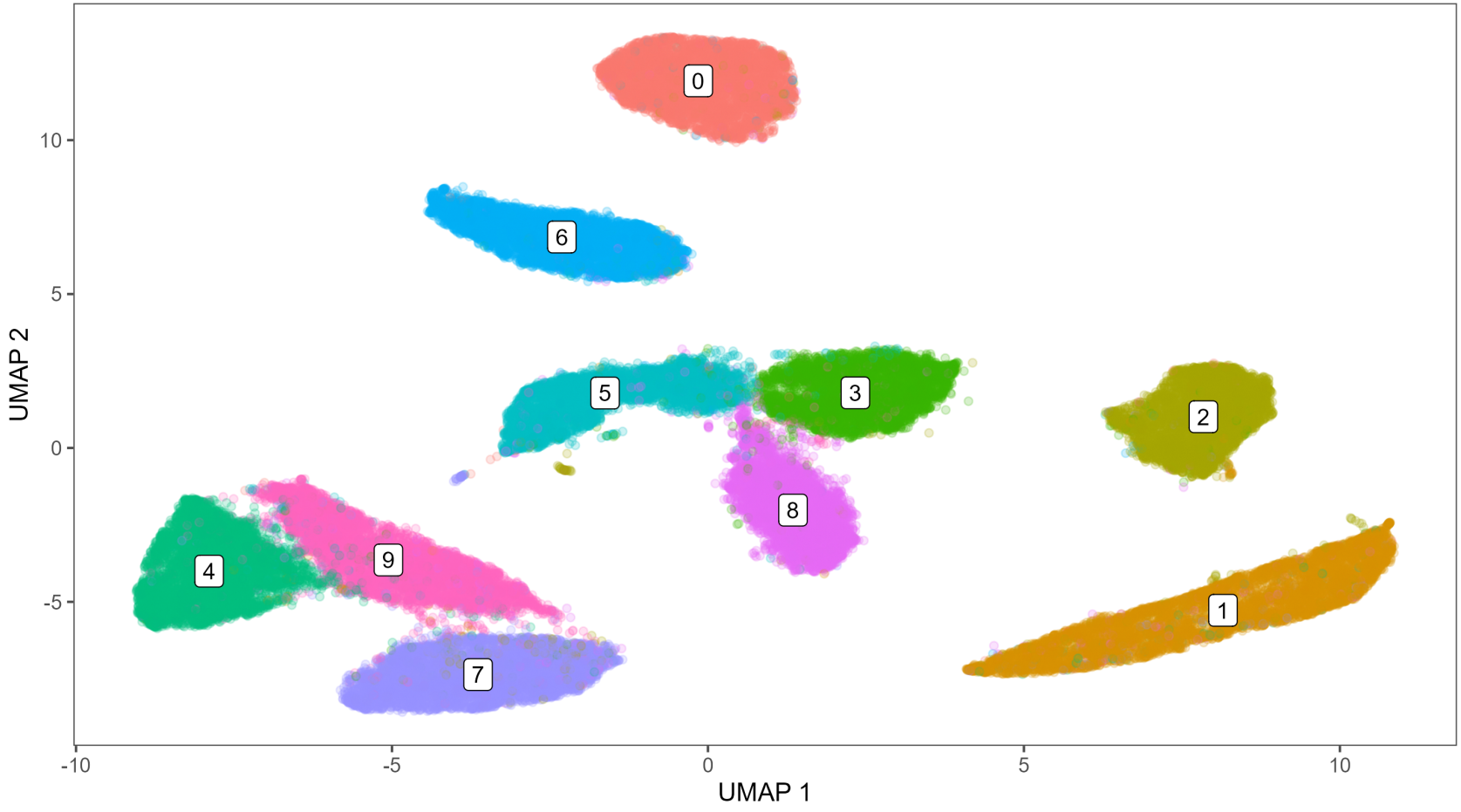

Figure 9: UMAP Plot of MNIST Digit Data

Batch Correction Evaluation Metrics

k-nearest neighbors batch effect test

- Constructed with the following premise in mind:
  - a subset of a well-mixed (i.e., batch corrected) dataset with batch effects removed should have the same distribution of batch labels as the full dataset
- kBET is a χ^2-based test for random subsets of the data for a fixed sized neighborhood around points
- Test is repeated for many points and results from each test (i.e., reject or failed to reject) are averaged to provide an overall rejection rate
  - If rejection rates are low, then the batch label distributions at the local level are similar to the global distribution

Batch Correction Evaluation Metrics

k-nearest neighbors batch effect test

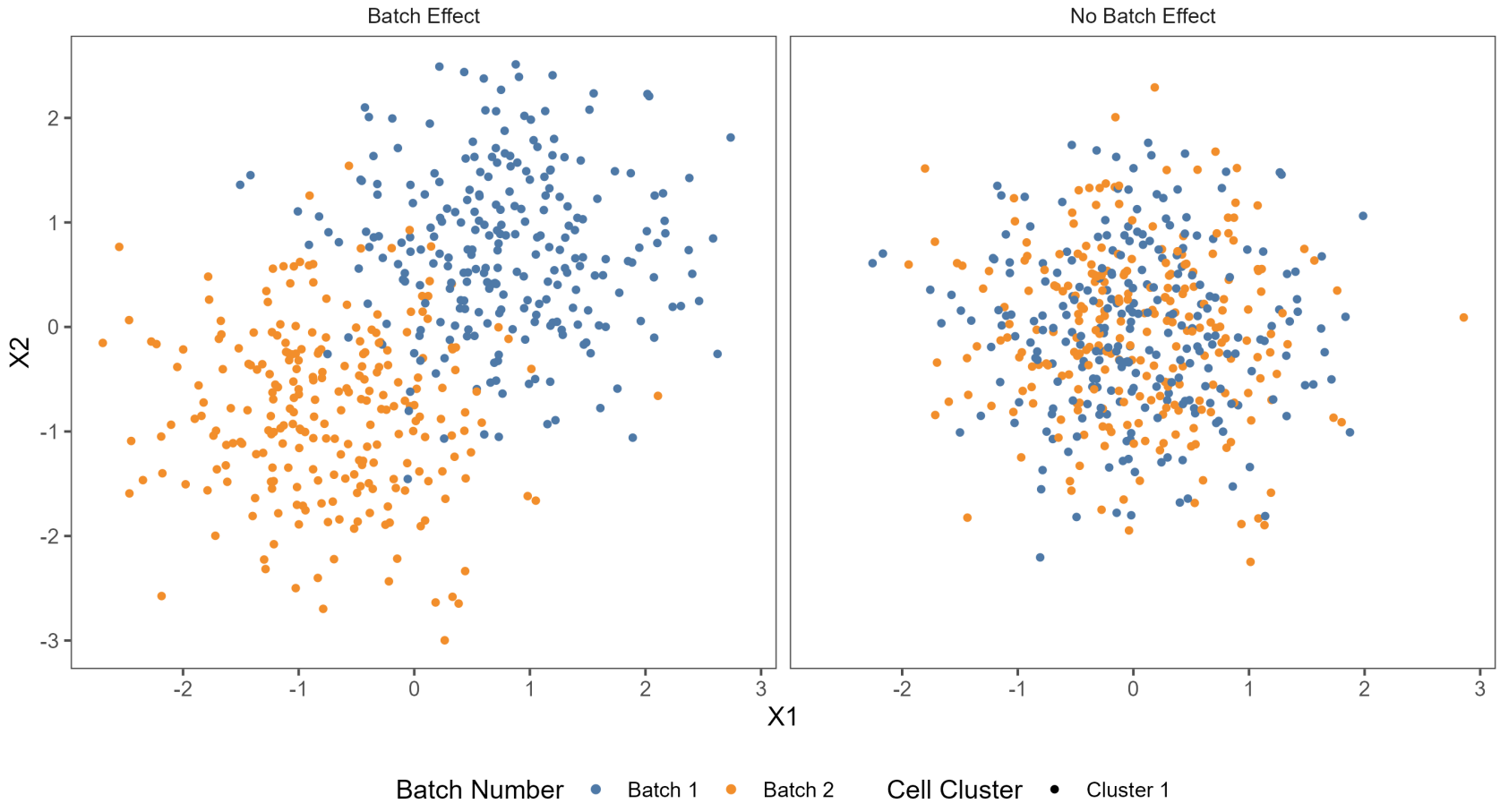

Figure 10: Toy dataset: cells from two batches

Batch Correction Evaluation Metrics

k-nearest neighbors batch effect test

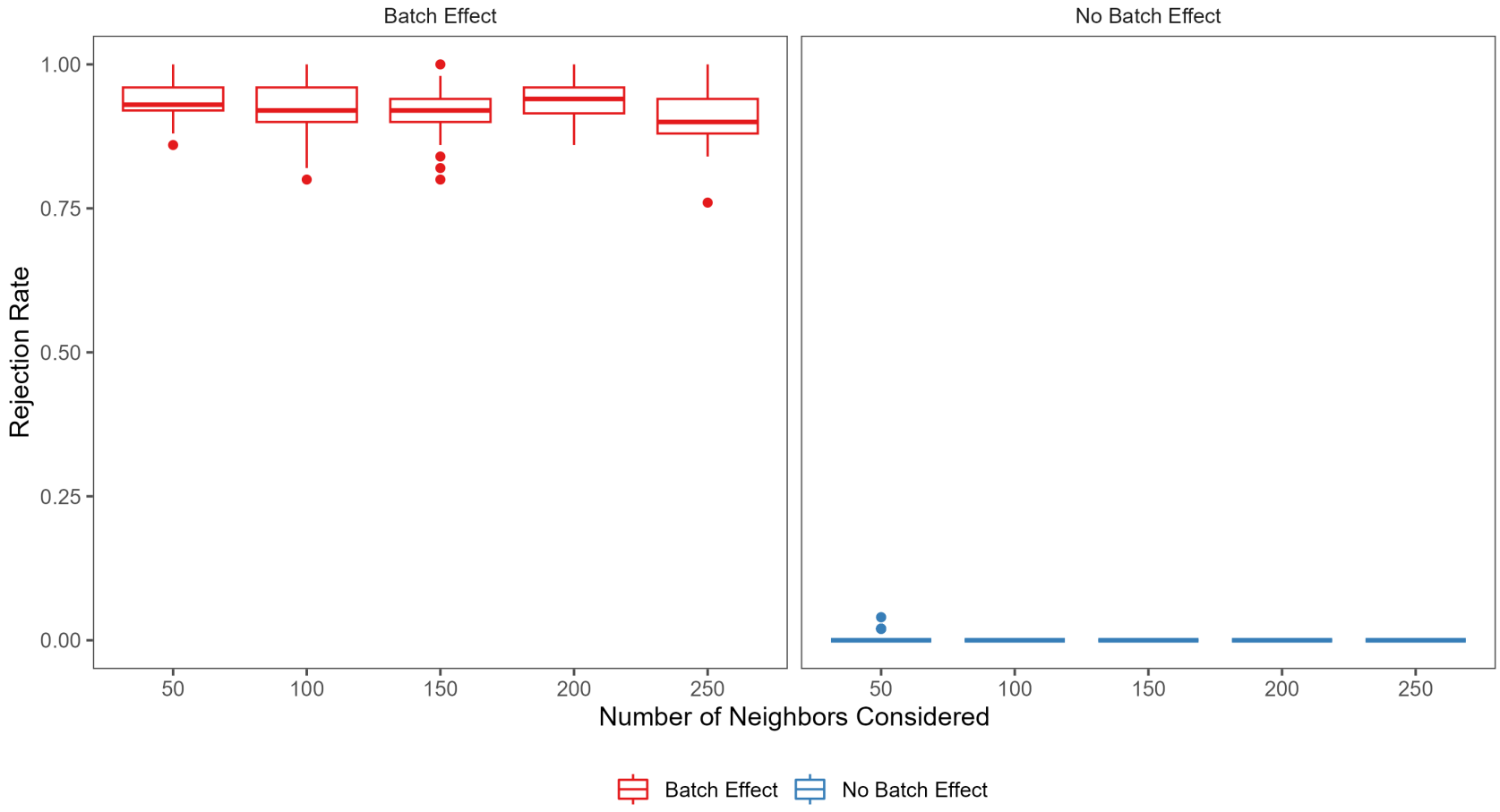

Figure 11: Rejection rates for toy dataset

Batch Correction Evaluation Metrics

k-nearest neighbors batch effect test

- We use the first 30 principal components from the batch-effect corrected datasets as input for kBET
- We calculate rejection rates for neighborhood sizes equal to 5%, 10%, 15%, 20%, and 25% of the number of cells being integrated
- We calculate the acceptance rate (1 - rejection rate) so that larger values are more desirable
- Acceptance rates for each method are then used for comparison

Batch Correction Evaluation Metrics

Average silhouette width

- Average silhouette width (ASW) is measure of consistency within clusters of a given dataset
- For each data point in a given cluster, we calculate the mean distance between itself and all other points in the same cluster
- We also calculate the smallest mean distance between itself and all other points not in the same cluster
- A silhouette is the difference between these two quantities scaled by the largest one
  - Takes on values between -1 and 1
  - Values close to 1 indicate a particular point is appropriately clustered
  - Values close to -1 indicate that a particular point may belong in a different cluster
  - Values close to 0 are borderline and could be in two or more clusters

Batch Correction Evaluation Metrics

Average silhouette width

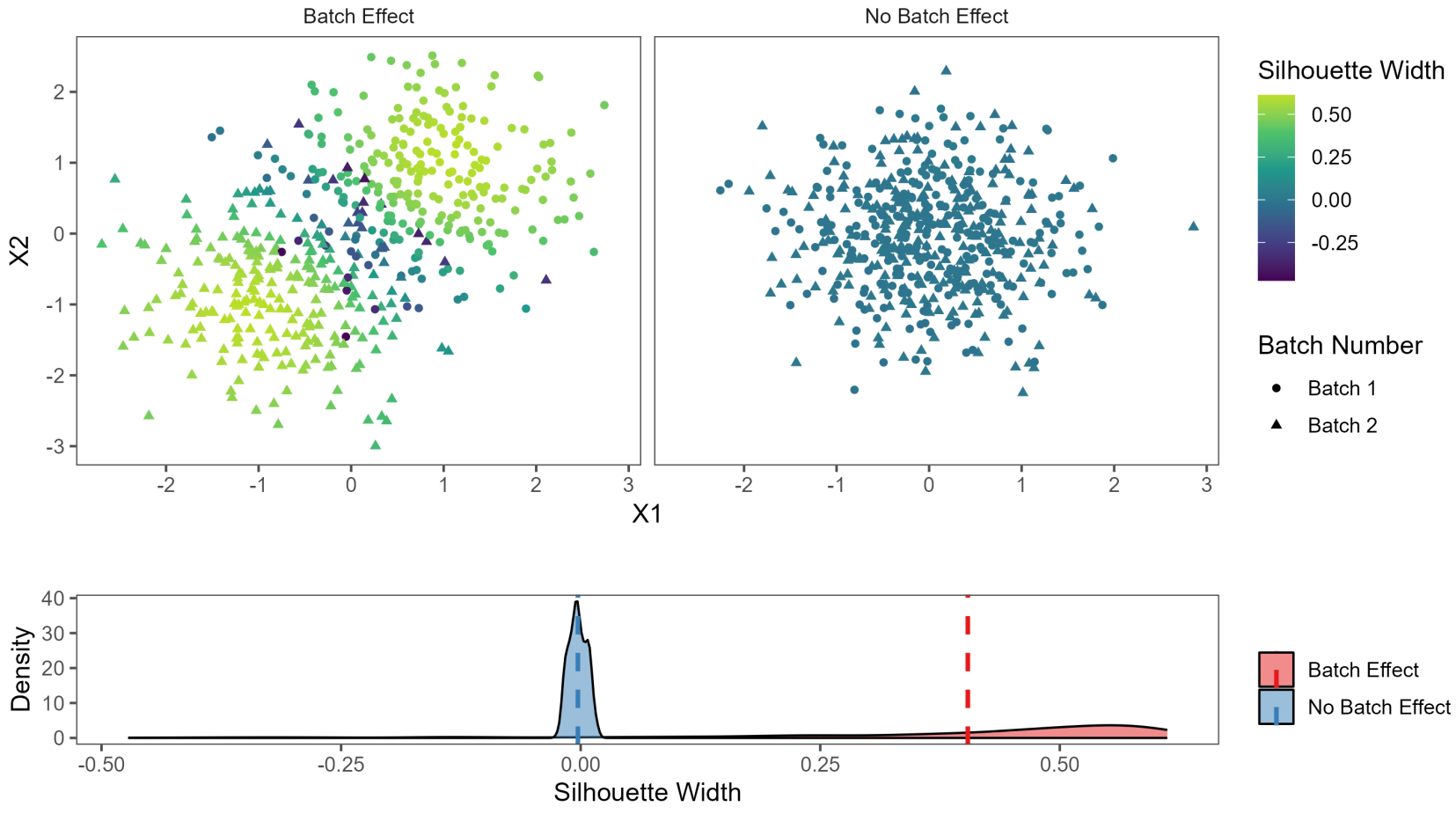

Figure 12: Silhouette widths for toy dataset

Batch Correction Evaluation Metrics

Average silhouette width

- We subsampled each dataset down to 80% of the original number of cells and used the first 30 principal components as input to calculate ASW
- This process is repeated 20 times for each method
- Two ASW metrics are calculated:
  - ASW batch, where we consider the batch labels as clusters
  - ASW cell type, where we consider the cell types as clusters
- Both of these metrics are separately scaled between 0 and 1
- We report 1 - scaled ASW batch so that larger values are more desirable
- The median ASW values for each method are used for comparison

Batch Correction Evaluation Metrics

Local inverse Simpson's index

- The local inverse Simpson's index begins by building local Gaussian kernel-based distributions of neighborhoods around each cell
- These probabilities are then used to calculate the inverse Simpson's index for a given cell
- These diversity scores correspond to the effective number of clusters in a particular cell's neighborhood

Batch Correction Evaluation Metrics

Simpson's index

- Used to measure the degree of concentration for individuals classified into types

$$\lambda = \sum_{i=1}^{R} p_i^2 \tag{7}$$

- λ is Simpson's index, R is the total number of types, and p_i is the proportion of individuals in class i
- $1/\lambda$ is the inverse Simpson's index

Batch Correction Evaluation Metrics

Inverse Simpson's index example

Table 4: Batch and Cell Type Frequency for Simulated Data

Class	n_i	p_i
A	80	0.267
B	125	0.417
C	95	0.317

- $\lambda = (\frac{80}{300})^2 + (\frac{125}{300})^2 + (\frac{95}{300})^2 = 0.345$
- $1/\lambda \approx 2.9$

Batch Correction Evaluation Metrics

Local inverse Simpson's index

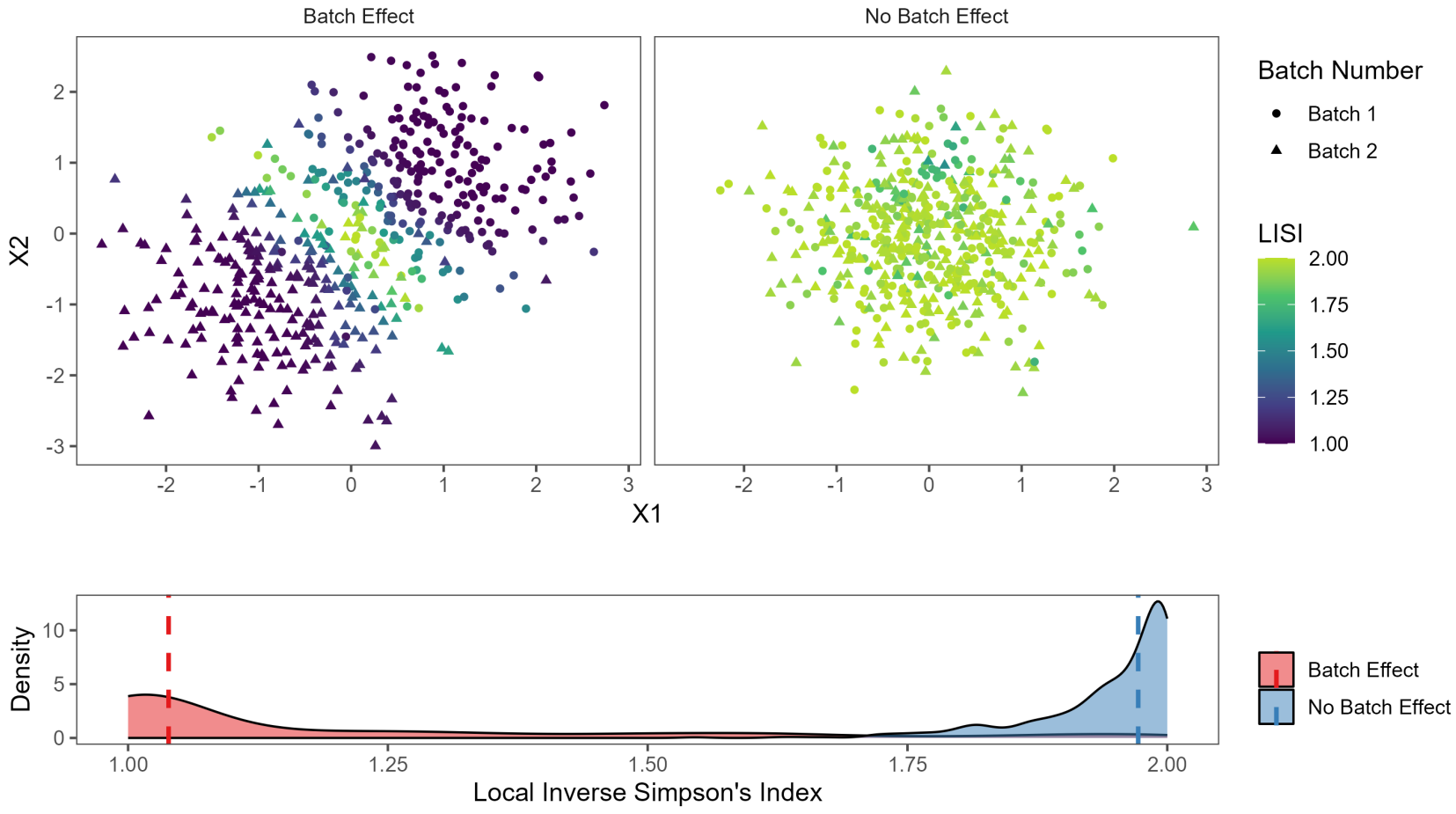

Figure 13: LISI scores for toy dataset

Batch Correction Evaluation Metrics

Local inverse Simpson's index

- We calculated two LISI metrics:
  - iLISI (integration LISI), where batch labels are considered the class
  - cLISI (cell type LISI), where cell type labels are considered the class
- Both LISI scores are calculated for each cell in the batch-effect corrected datasets for each method
- Both LISI scores are separately scaled between 0 and 1
- We report 1 - scaled cLISI so that large values are more desirable
- The median LISI values for each method are used for comparison

Results

JIVE Runtime Improvements

Partial SVD

- The original `svd()` function R performs the decompositions in approximately 1.67 seconds on average
- The partial SVD function `svds()` from the `RSpectra` R package performed them in 0.08, 0.12, and 0.15 seconds on average
  - 95.4%, 93.1%, and 91.0% shorter runtime when estimating $k = 1$, 5, and 10 singular values/vectors, respectively
  - Important as a SVD is calculated for the joint structure matrix and each individual structure matrix in every iteration of the JIVE estimation algorithm

JIVE Runtime Improvements

Matrix multiplication

- The base R %*% operator and the function from the Armadillo C++ libary both averaged a runtime of just under 0.3 seconds
- The four Eigen C++ library functions utilizing 1, 2, 4, and 8 CPU cores had average runtimes of 0.093, 0.047, 0.027, and 0.02 seconds on average, respectively
  - 67.5%, 83.2%, 90.6%, and 92.9% shorter runtimes when using the Eigen functions with one, two, four, and eight CPU cores, respectively

JIVE Runtime Improvements

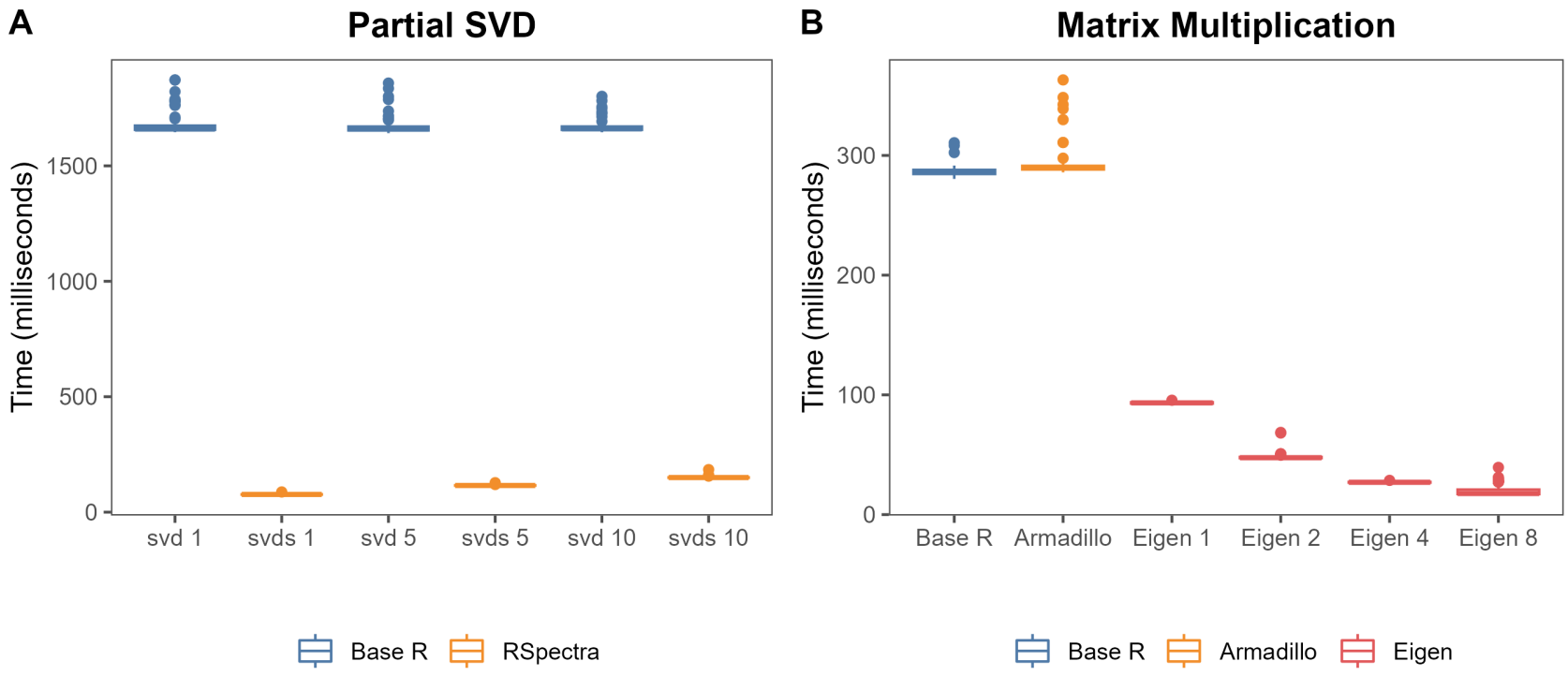

Figure 14: Benchmarks for partial SVD and matrix multiplication

JIVE Runtime Improvements

Overall improvements

- The original `R.JIVE` functions perform the decomposition on the two matrices A and B in about 35.8 seconds on average
- The updated functions completes the decomposition in about 4.1 seconds on average
- Outputs were nearly identical, as seen by the table below comparing their proportions of variances

Table 5: Proportion of Variance Attributed to JIVE Decomposition

Original	Data 1	Data 2	Updated	Data 1	Data 2
Joint	0.346	0.161	Joint	0.346	0.161
Individual	0.400	0.582	Individual	0.400	0.582
Residual	0.254	0.256	Residual	0.254	0.256

JIVE Runtime Improvements

Overall improvements

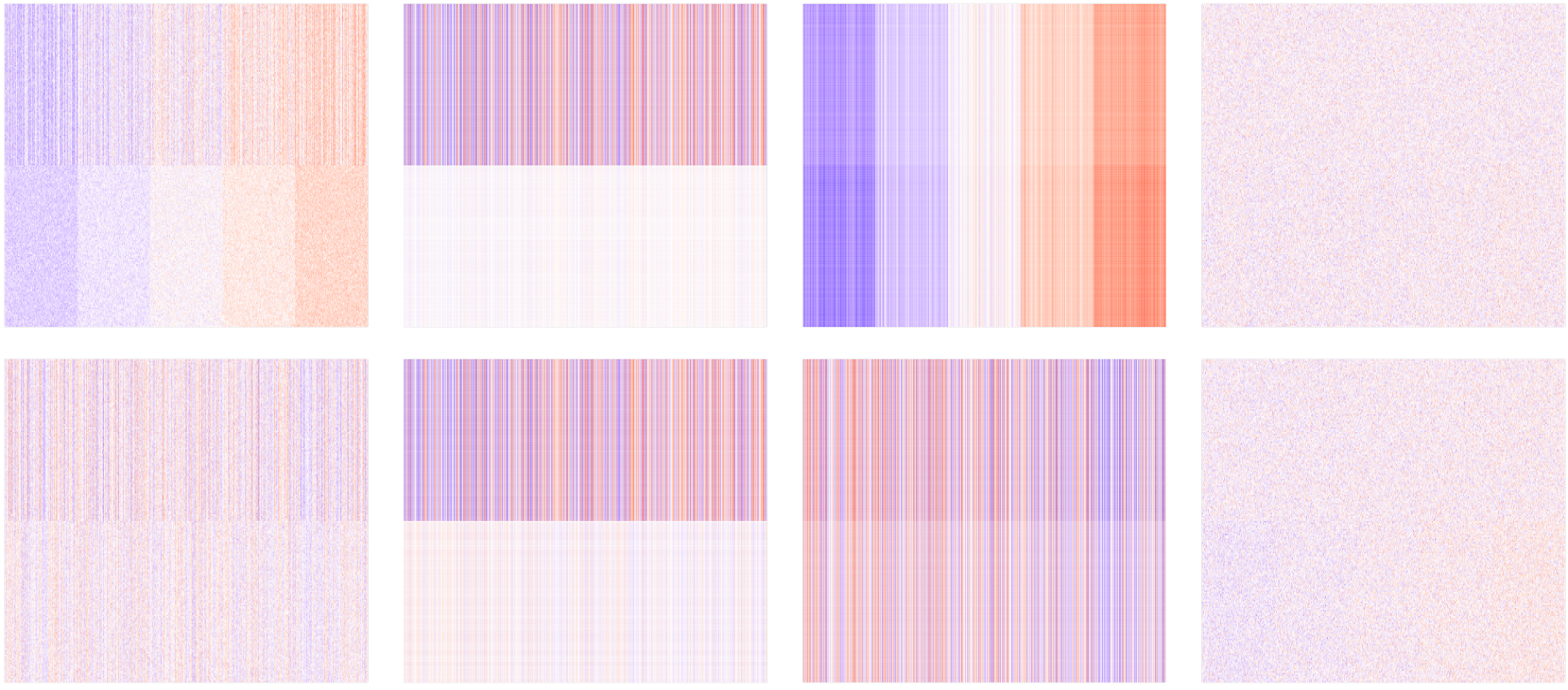

Figure 15: Updated JIVE decomposition output

JIVE Runtime Improvements

Overall improvements

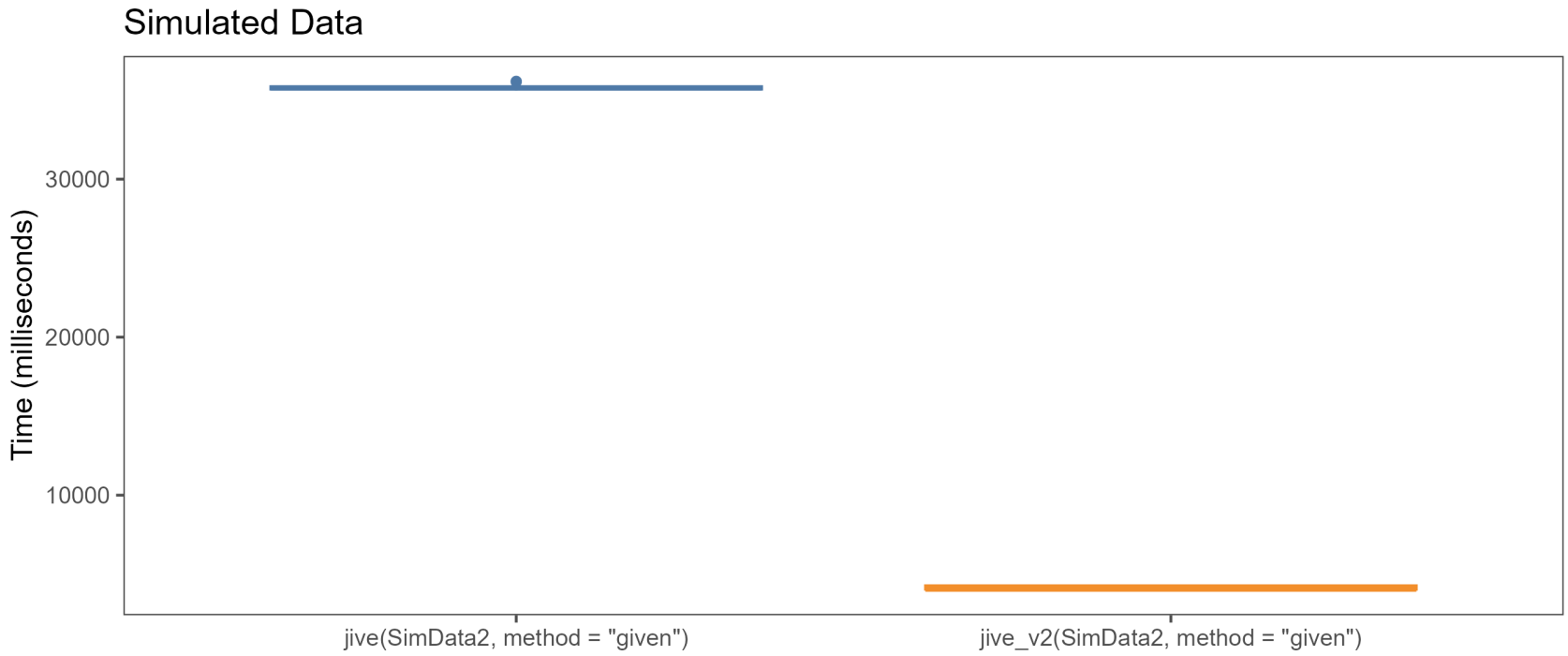

Figure 16: JIVE function benchmarks

Simulated Data

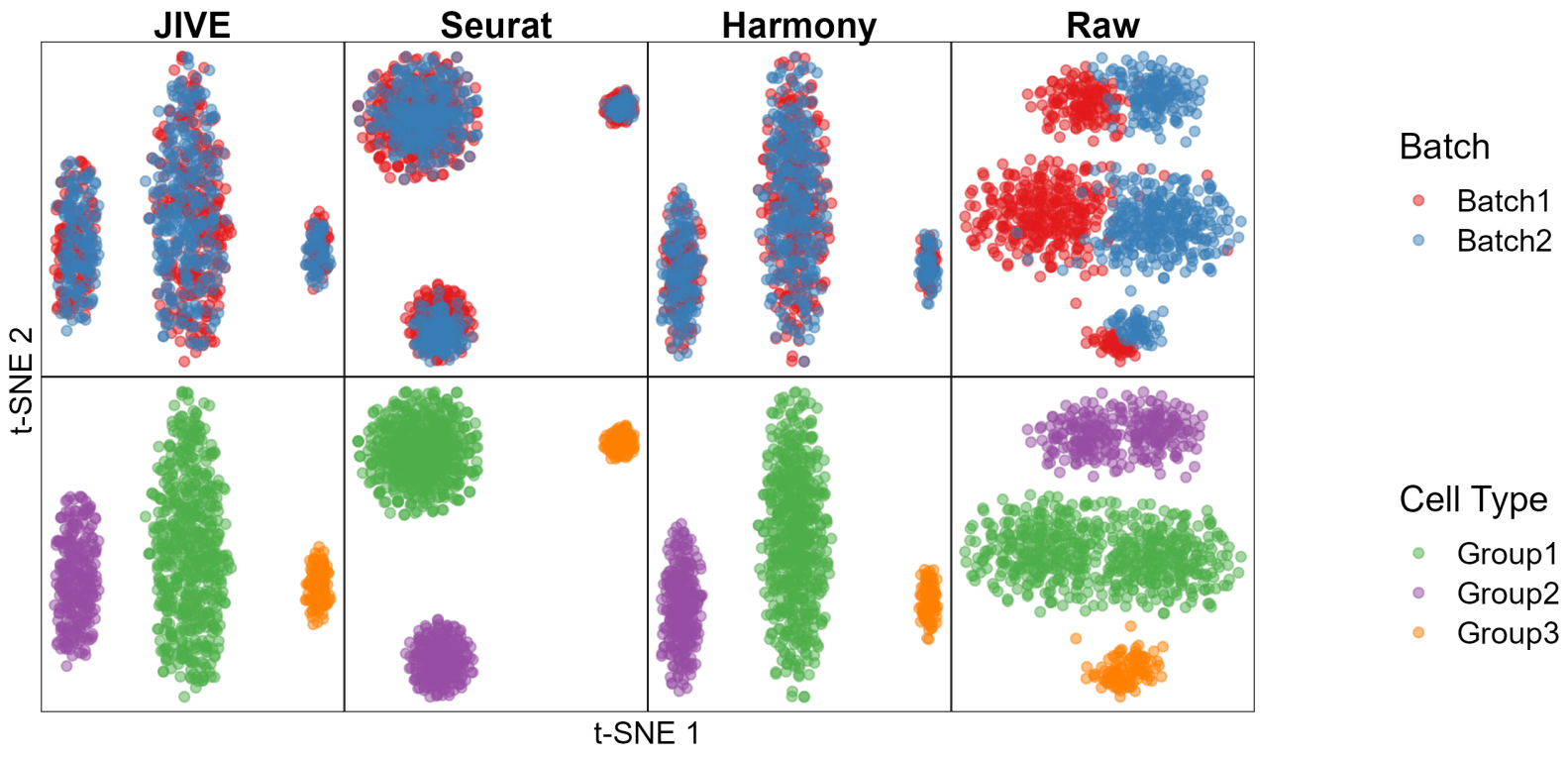

Figure 17: t-SNE plot for simulated data

Simulated Data

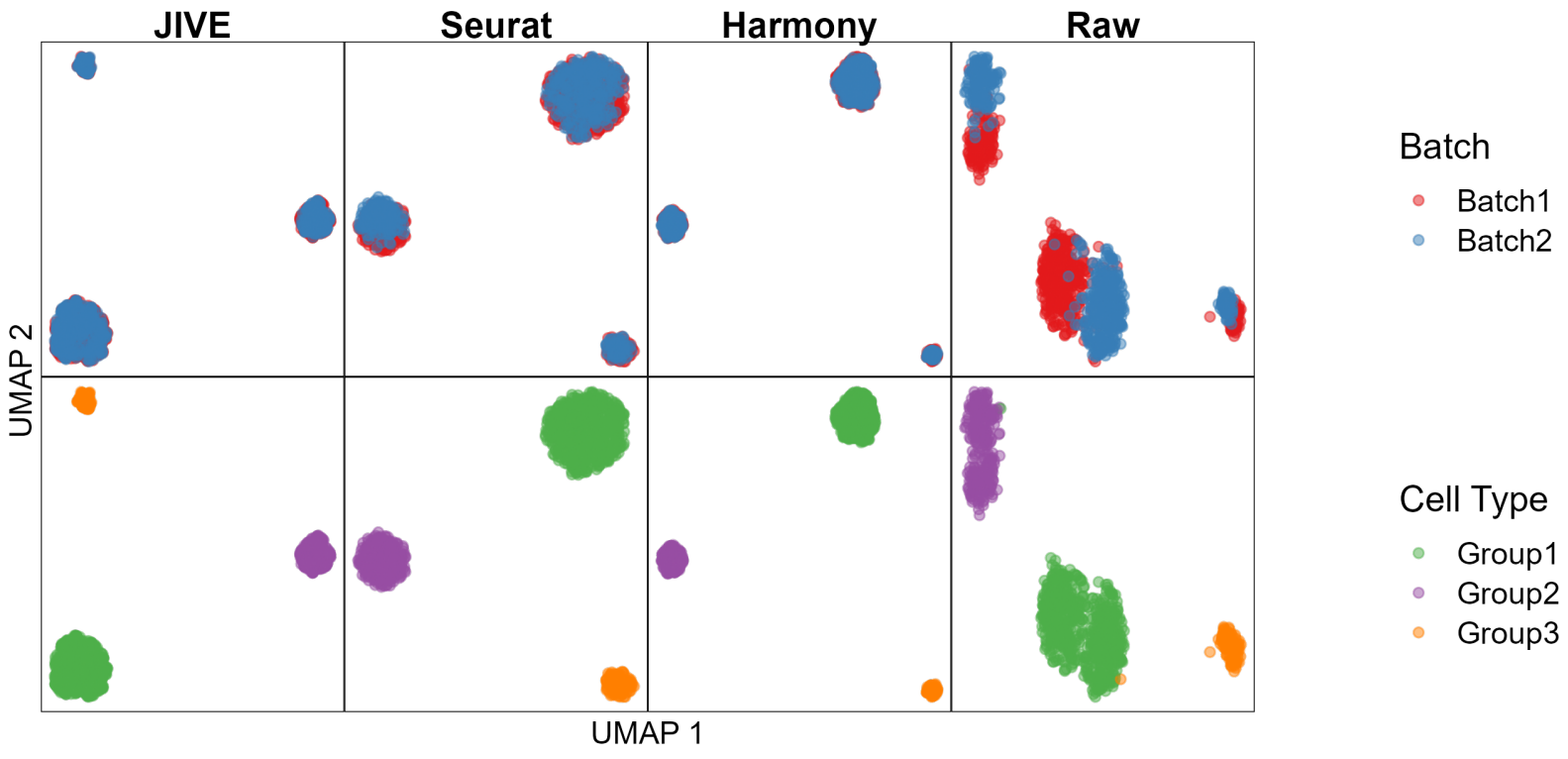

Figure 18: UMAP plot for simulated data

Simulated Data

Quantitative metrics

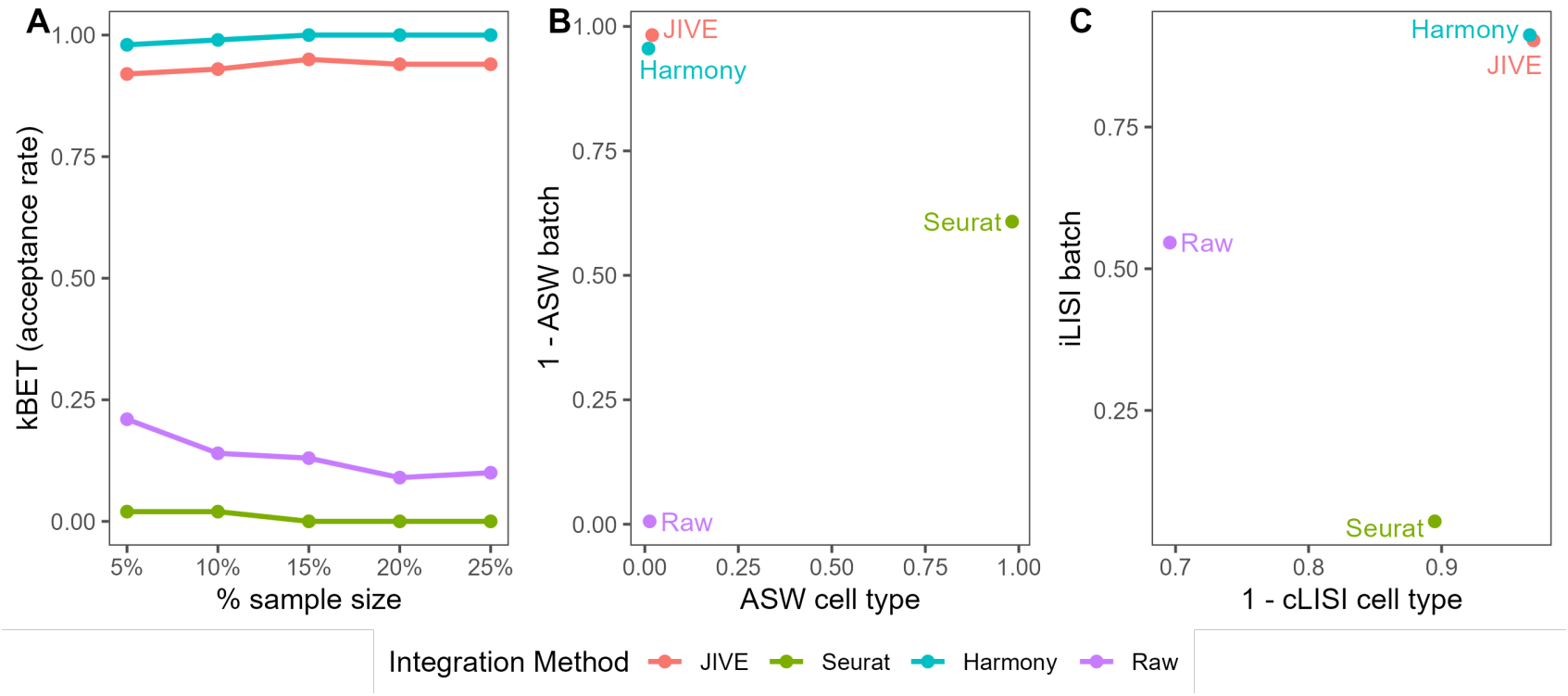

Figure 19: (A) kBET acceptance rates, (B) ASW metrics, (C) LISI metrics for simulated data

Bacher T-Cell Data

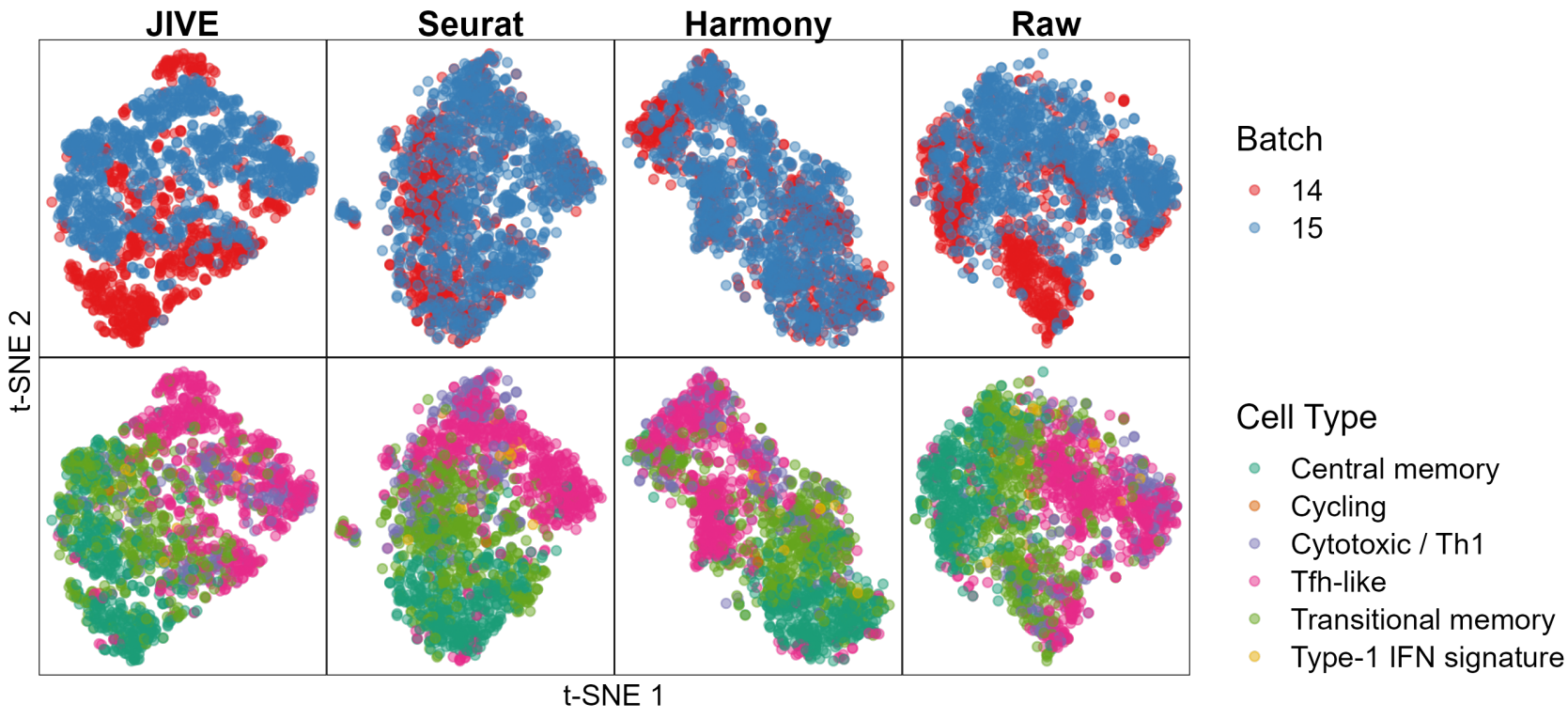

Figure 20: t-SNE plot for Bacher T-cell data

Bacher T-Cell Data

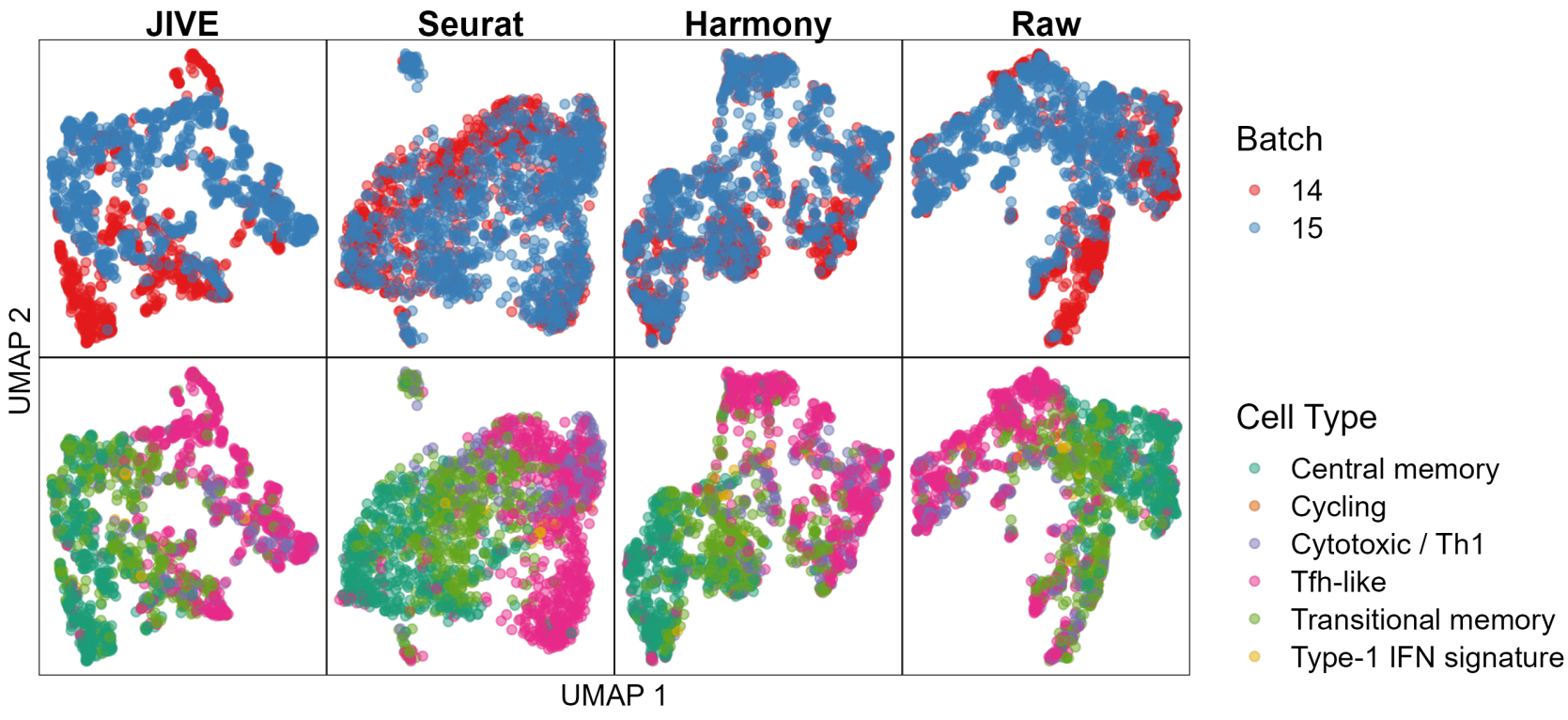

Figure 21: UMAP plot for Bacher T-cell data

Bacher T-Cell Data

Quantitative metrics

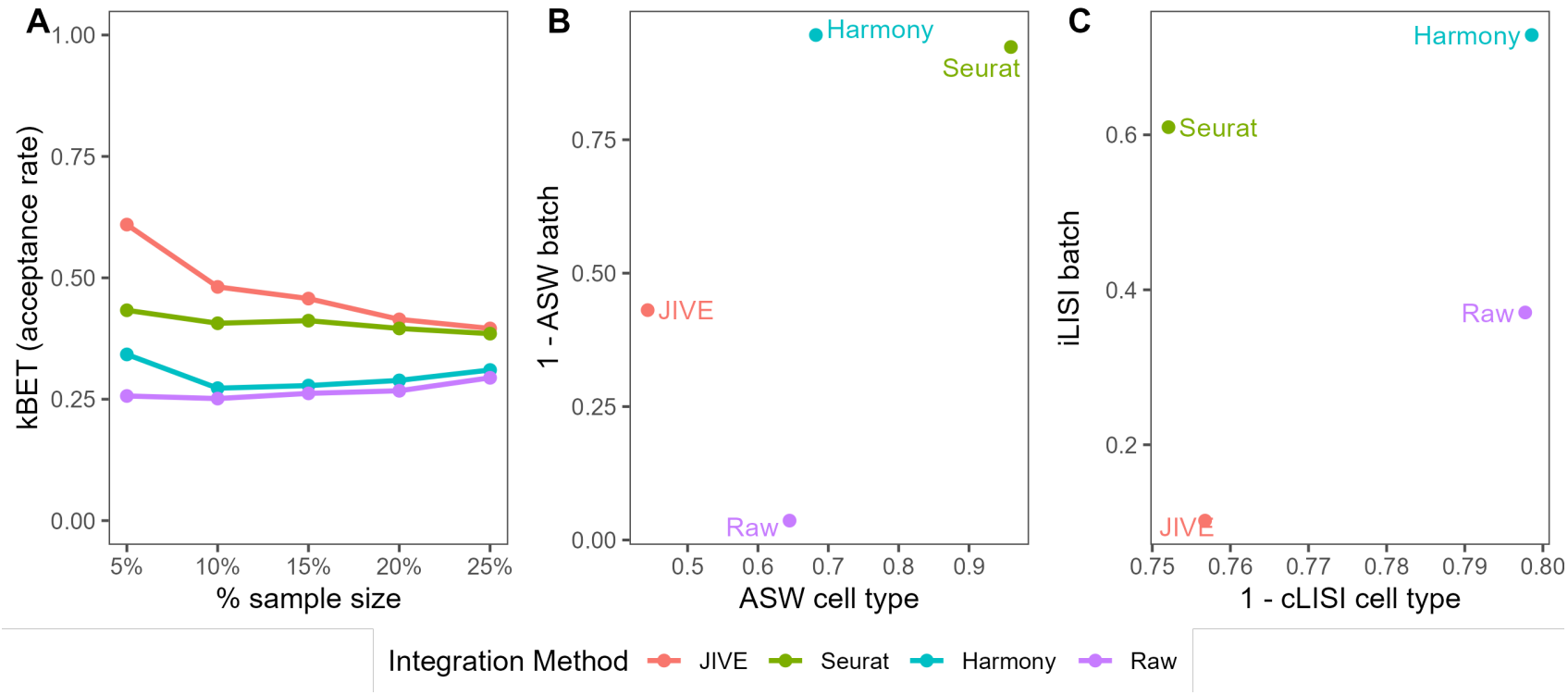

Figure 22: (A) kBET acceptance rates, (B) ASW metrics, (C) LISI metrics for Bacher T-cell data

Zilionis Mouse Lung Data

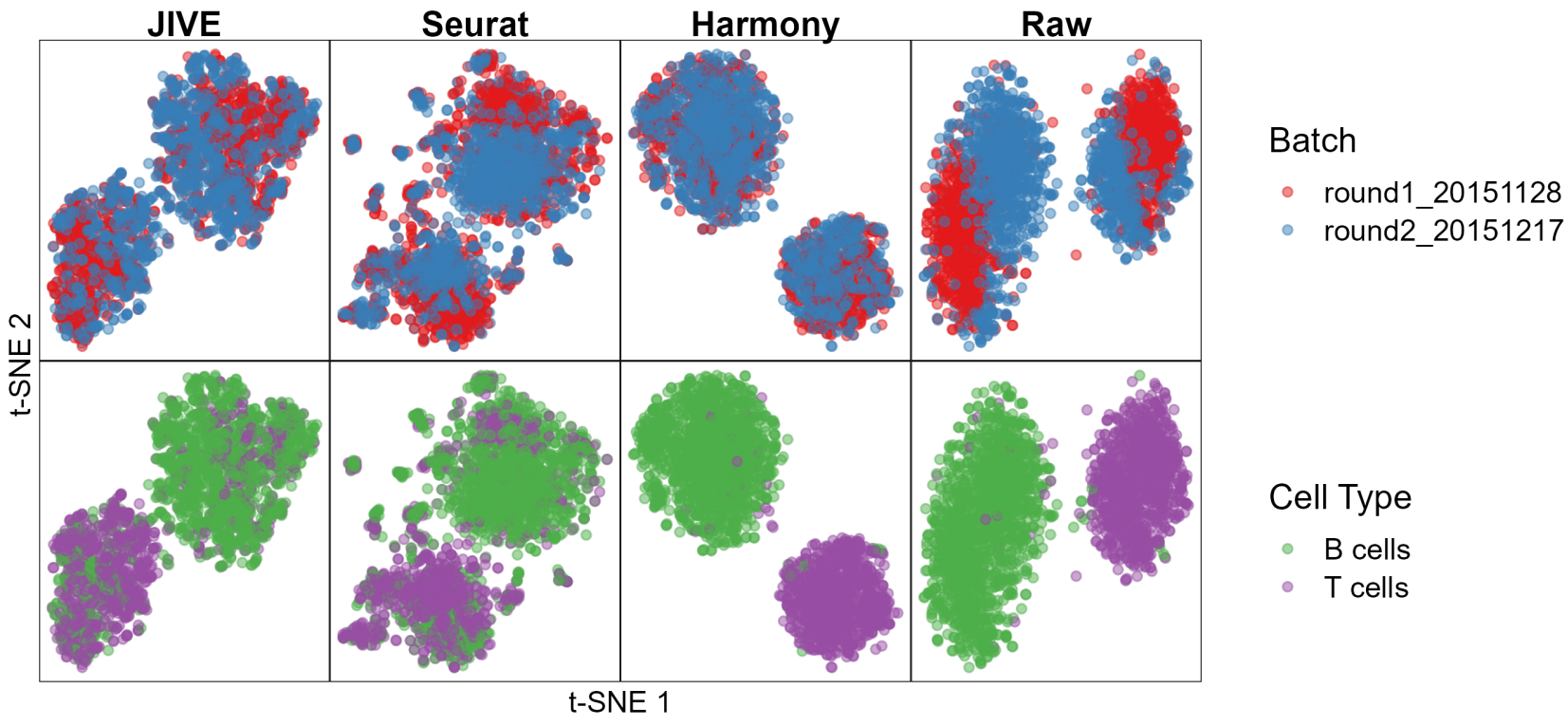

Figure 23: t-SNE plot for Zilionis mouse lung data

Zilionis Mouse Lung Data

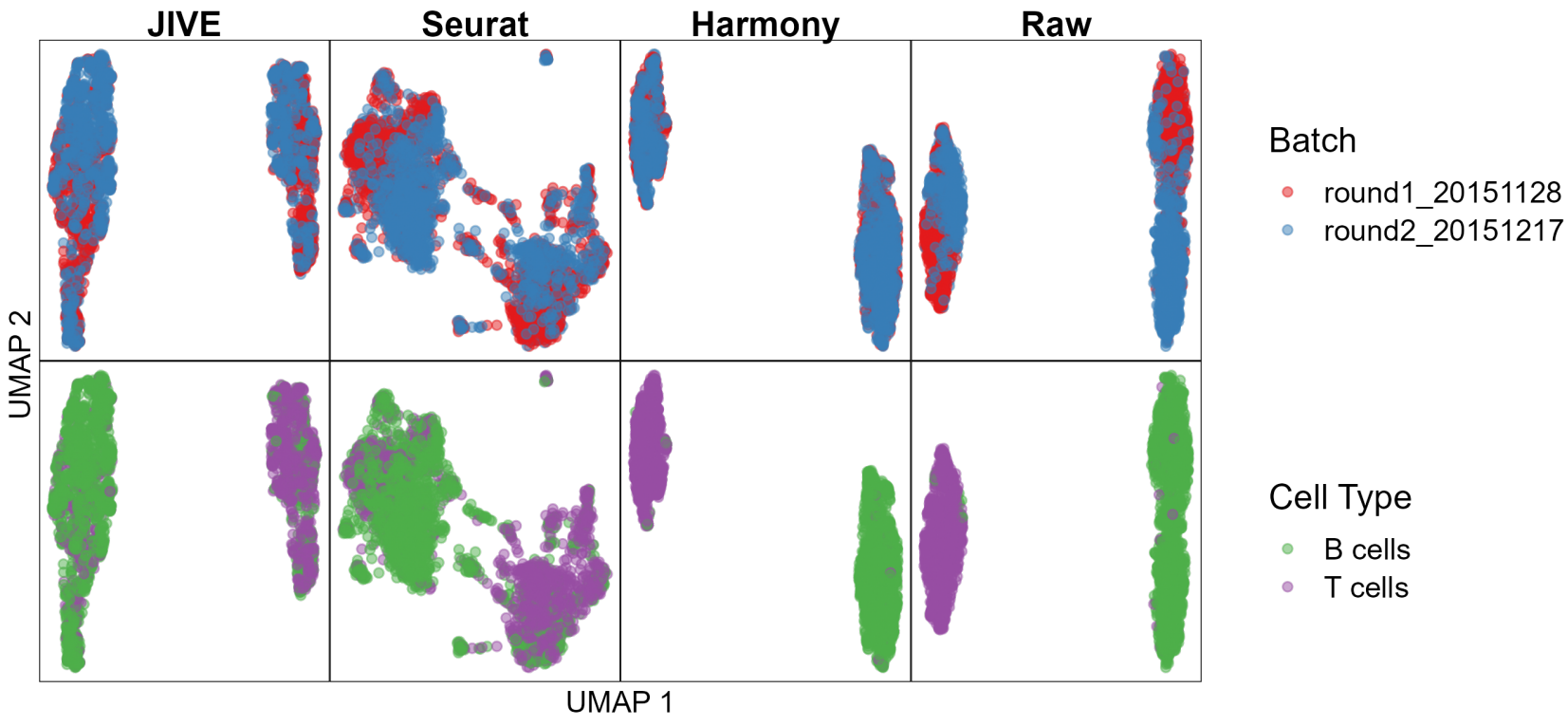

Figure 24: UMAP plot for Zilionis mouse lung data

Zilionis Mouse Lung Data

Quantitative metrics

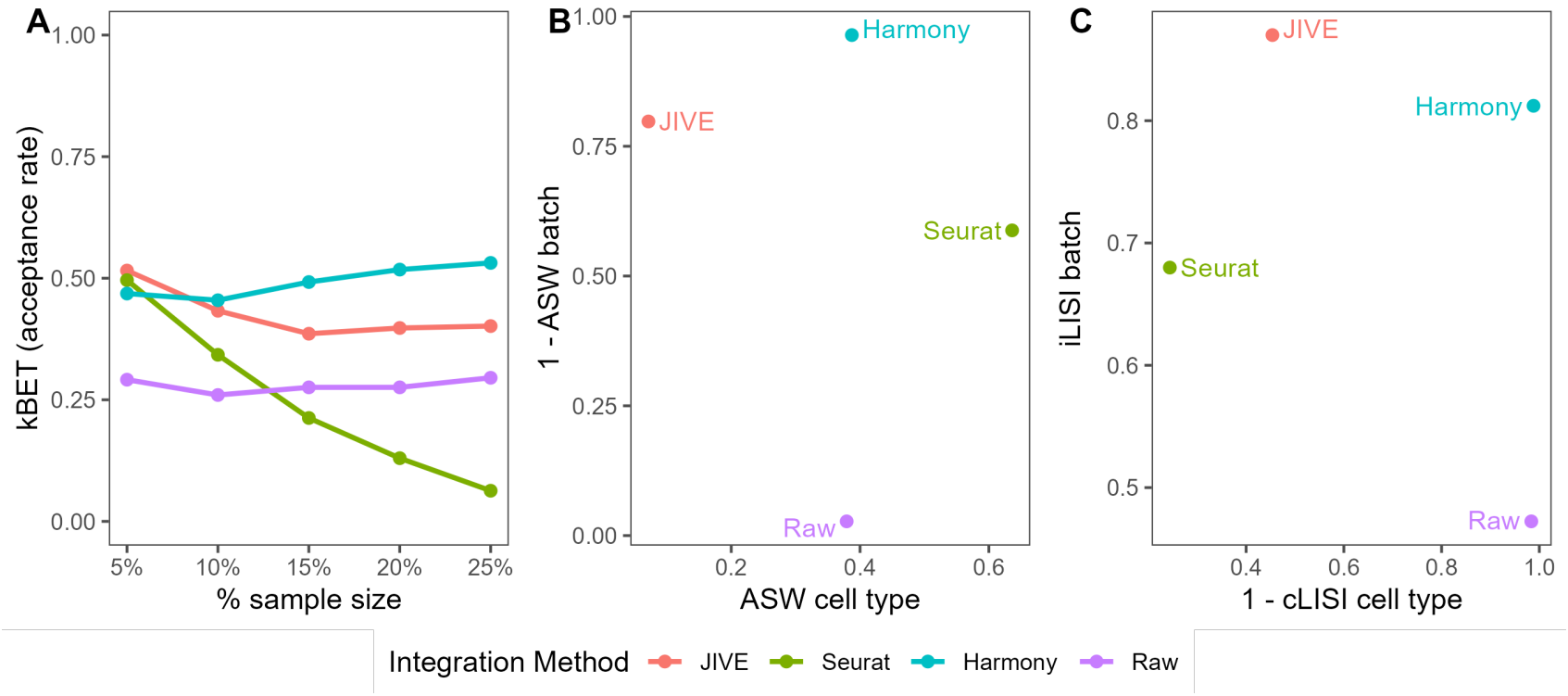

Figure 25: (A) kBET acceptance rates, (B) ASW metrics, (C) LISI metrics for Bacher T-cell data

Discussion

Method performance

Harmony

- Performed the best in the simulated data and Zilionis mouse lung data
- Consistently performed well on kBET and LISI metrics

JIVE

- Performed second best with regards to metrics concerning batch mixing, struggled with cell type purity metrics
- Able to keep up with Harmony in both the simulated data and the Zilionis mouse lung data

Seurat

- Performed second best at metrics concerning cell type purity and had its best performance in the nebulous Bacher T-cell data
- Poor kBET performance in the simulated data seemed to contradict the t-SNE and UMAP plots

Method performance