Scedar: a scalable Python package for single-cell RNA-seq exploratory data analysis

Yuanchao Zhang*[1,2] and Deanne M. Taylor†[1,3]

[1]Department of Biomedical and Health Informatics, The Children's Hospital of Philadelphia, Philadelphia, PA 19041, USA
[2]Department of Genetics, Rutgers University, Piscataway, NJ 08854, USA
[3]Department of Pediatrics, Perelman School of Medicine, University of Pennsylvania, Philadelphia, Pennsylvania, 19104, USA

Abstract

Motivation In single-cell RNA-seq (scRNA-seq) experiments, the number of individual cells has increased exponentially due to significant improvements on single-cell isolation and massively parallel sequencing technologies. However, computational methods have not scaled to the same order, presenting analytical challenges to the increasing data size.

Results We present `scedar`, a scalable Python package for scRNA-seq exploratory data analysis. The package provides a convenient and reliable interface for efficiently performing quality control, visualization and clustering on large-scale scRNA-seq datasets. The high-performance data structures representing scRNA-seq data in the package are extensible and modular, which could be reused to develop other analytical methods and refactored to support analysis at larger scale.

Availability The open source package is distributed under the terms of MIT license at https://pypi.org/project/scedar.

1 Introduction

Cost-effective large scale transcriptomic profiling of individual cells is enabled by the development of microfluidic, nanodroplet, and massively parallel sequencing technologies. Using these technologies, single-cell RNA-seq (scRNA-seq) experiments usually generate transcriptomic profiles of thousands to millions of individual cells (Svensson *et al.*, 2018; Rozenblatt-Rosen *et al.*, 2017), which are represented by a large matrix with each entry storing the number of sequencing reads mapped to a corresponding gene (column) within its corresponding cell (row). As the number of annotated transcripts can exceed fifty thousand in both human and mouse genomes (Harrow *et al.*, 2012), matrix sizes can potentially reach upwards to many billions of entries.

However, previous computational methods are not designed for scalability (Kiselev *et al.*, 2017; Vallejos *et al.*, 2015; Aibar *et al.*, 2017; žurauskienė and Yau, 2016; Zeisel *et al.*, 2015). Although they all have outstanding analytical performance on specific datasets, they usually require multiple

*yuanchao.zhang@rutgers.edu
†taylordm@email.chop.edu

trials with different parameters and pre-processing procedures to obtain the optimal performance on any new dataset, which could be time-consuming or even infeasible on large datasets.

In addition, previous computational methods usually disregard standard software engineering practices that are important to analyze large scale datasets, especially for error and exception handling, software testing, and memory management. Proper error handling ensures that any part of the analysis could fail fast, safe and informative if required, which helps to prevent analysis or whole system failure without appropriate error messaging. Responsive exception handling also allows safe interruption at any point in the calculation. Including comprehensive software testing routines into a package is helpful not only for debugging during initial development, but also importantly supports package extension in the open software phase. Although garbage collection is a common feature in modern programming languages, an efficient memory management system copies data only when necessary and caches frequently accessed intermediate results, which could significantly improve the time and space efficiency especially in large-scale parallelized data analysis.

In order to facilitate the exploration of large-scale scRNA-seq datasets, we developed a scalable and reliable Python package, single-cell exploratory data analysis for RNA-seq (`scedar`). `Scedar` implements the analytical routines as three modules: quality control (QC), exploratory data analysis (EDA), and clustering. The QC module detects cell outliers and impute gene dropouts based on the k-nearest neighbors of each cell. The EDA module provides data structures specifically designed for scRNA-seq data that are efficient, parallel, and modular for fast visualization and analytical processing. The clustering module provides a novel cell clustering algorithm named MIRAC (see Algorithm 1), minimum description length (MDL) iteratively regularized agglomerative clustering. The clustering module can also identify genes that are able to distinguish the selected clusters using a sparsity-aware gradient boosting tree system, XGBoost (Chen and Guestrin, 2016).

2 Methods

2.1 `scedar` package design

`Scedar` aims to facilitate the development of scalable statistical models of scRNA-seq data by efficient EDA. Although we also provide analytical methods for various purposes, they are designed to facilitate data exploration, rather than surpassing the analytical performances of numerous other methods.

We designed `scedar` in an object-oriented manner for quickly exploring large-scale scRNA-seq transcription matrices on a remote server utilizing parallel computing techniques. The application programming interface (API) is designed to be intuitive to users familiar with the R programming language. The standard analysis workflow is to explore the dataset, cluster cells, and identify cluster separating genes (Figure 1), which is implemented as four main modules: data structure, quality control, clustering and visualization.

The core data structure stores each transcription level matrix as a standalone instance, and it can easily be extended to support customized analytical procedures. The built-in extensions include common procedures like pairwise distance computation, Principal Component Analysis (PCA) (Pearson, 1901), t-Distributed Stochastic Neighbor Embedding (t-SNE) (van der Maaten and Hinton, 2008), UMAP (McInnes and Healy, 2018), and k-nearest neighbor graph (Jacomy *et al.*, 2014). We optimized time and memory efficiency with the following design patterns: parallel computing, lazy loading, caching and copy-on-write.

The quality control and clustering modules utilize the data structure and parallel computing to efficiently perform analytical procedures, and the results are stored in the data structure for further reference.

The visualization module contains plotting methods optimized for large datasets, especially for plotting heatmaps. For example, it takes less than two minutes to generate a heatmap image from a $50,000 \times 20,000$ matrix containing random standard normal entries.

2.2 `scedar` package development

`Scedar` is built upon various high-performance scientific computing and visualization packages. `Scedar` is also extensively benchmarked and tested by unit testing, with comprehensive coverage on statements and branches.

`Scedar` uses the following packages:

- `numpy` (Travis E. Oliphant, 2006) for matrix representation and operations.
- `scipy` (Virtanen *et al.*, 2018) for fast Gaussian kernel density estimation, hierarchical clustering and sparse matrix.
- `matplotlib` (Hunter, 2007) and `seaborn` (Waskom *et al.*, 2017) for visualization.
- `pandas` (McKinney, 2010) for data frame representation.
- `scikit-learn` (Pedregosa *et al.*, 2011) for parallel computation of pairwise distances, k-nearest neighbor data structure, PCA and t-SNE.
- `XGBoost` (Chen and Guestrin, 2016) for scalable gradient boosting tree.
- `networkx` (Hagberg *et al.*, 2008) for graph data structure and visualization.
- `ForceAtlas2` (Jacomy *et al.*, 2014) for scalable force-directed graph layout.

We use the Python package `pytest` as the unit testing framework to ensure that `scedar` has expected behaviors. We tested each member of the package with multiple testing environments, in order to make sure that all statements and branches are executed in the tests, i.e. comprehensive code coverage. The code coverage is measured by the Python package `coverage`. Although comprehensive testing coverage does not guarantee that the package is bug-free, it eliminates obvious errors, e.g. accessing local variables before definition.

Comprehensive unit testing greatly helps with validating correctness, ensuring reproducibility and refactoring the code. We carefully tested our analytical procedures with multiple input datasets to cover standard and edge cases, in order to make sure that the results are correct and reproducible. We also confidently refactored the code multiple times throughout the development process to improve backend performance, accommodate special use cases, and reorganize intra-package dependencies. For a non-trivial package with multiple inter-related components like `scedar`, changes in certain components may unexpectedly affect other components that directly or indirectly use the changed ones, so that validating the correctness after the changes requires a significant amount of effort without comprehensive tests.

2.3 Minimum description length iteratively regulated agglomerative clustering

Minimum description length (MDL) iteratively regulated agglomerative clustering (MIRAC) extends hierarchical agglomerative clustering (HAC) (Müllner, 2011) in a divide and conquer manner for scRNA-seq data. Input with raw or dimensionality reduced scRNA-seq data, MIRAC starts with one round of bottom-up HAC to build a tree with optimal linear leaf ordering (Bar-Joseph *et al.*,

2001), and the tree is then divided into small sub-clusters, which are further merged iteratively into clusters. Because each individual cluster becomes more homogenous with higher number of clusters, the iterative merging process is regularized with the MDL principle (Hansen and Yu, 2001). The time complexity of MIRAC is $O(n^4 + mn^2)$, where n is the number of samples, and m is the number of features. The space complexity is $O(n^2 + mn)$. Relevant mathematical theories and notations of MDL are briefly described in Supplementary Method section 1.1 . The pseudo-code of MIRAC is shown in Algorithm 1.

2.3.1 Rationale

The rationale behind MIRAC is to reduce the number of cell partitions that need to be evaluated and find a good one among them.

The number of all possible partitions of a set with n elements is the Bell number, which could be computed with the following recurrence $B_n = \sum_{i=0}^{n-1} B_k \binom{n-1}{k}$, where $n \geq 1$ and $B_0 = 1$ (Wilf, 1999, chap. 1). It is computationally intractable to compute the code lengths of B_n partitions, so we reduced the number of cell partitions to evaluate with the following steps:

1. We only evaluate the partitions of n cells that consecutively divide the optimal HAC tree T leaf ordering. The HAC tree with optimal leaf ordering maximizes the sum of similarity measurements between adjacent items in the leaf ordering (Bar-Joseph *et al.*, 2001), which could be computed with an algorithm developed by Bar-Joseph *et al.* (2001) that runs in time $O(n^4)$. The number of partitions that consecutively divide an ordering is the same as the number of compositions of n, which is $C_n = 2^{n-1}$. Because this number still grows exponentially with regard to n, it is necessary to further reduce the set of partitions for evaluation.

2. Within step 1 partitions, we only evaluate those with all clusters having $\geq t$ cells. The number of such partitions is the same as the compositions of n with all summands $\geq t$, which is

$$C_n^{\{t,t+1,\ldots,n\}} = \sum_{k=1}^{\lfloor n/t \rfloor} \binom{n - k(t-1) - 1}{k-1}$$

given by Abramson (1976). The growth rate of $C_n^{\{t,t+1,\ldots,n\}}$ with regard to n is smaller, but we still need to reduce it further. For example, $C_n^{\{20,21,\ldots,n\}}$ for n in $\langle$ 20, 40, 60, 80, 100, 120, 140, 160, 180, 200 $\rangle$ are $\langle$ 1, 2, 23, 274, 2695, 24941, 232016, 2184520, 20628613, 194570810 $\rangle$, and the values of 2^n are $\langle$ 1048576, 1099511627776, ..., 1.607×10^{60} $\rangle$.

3. Within step 2 partitions, we only evaluate the ones that could be generated by merging adjacent clusters of a specific step 1 partition $P^s = \{P_1^s, P_2^s, \ldots, P_k^s\}$. We generate P^s by recursively divide the root HAC tree T until the partitioned subtree has $\leq t-1$ leaves. Thus, $\lceil n/(t-1) \rceil \leq k \leq n$. Let $M(P^s)$ denote the number of step 2 partitions that could be generated by merging adjacent clusters of P^s. The upper bound of $M(P^s)$ is the same as the number of step 2 partitions, which could be reached when $k = n$. The lower bound of $M(P^s)$ is not straightforward, since the merged partitions should have all clusters with $\geq t$ cells.

In order to find a good cell partition P^g in the subset of all possible ones, we iteratively inspect each cluster of P^s and merge it with either its left or right adjacent cluster according to the similarity determined by the two-stage MDL scheme described above. The procedure has the following steps:

1. Merge P^s clusters in the beginning of the optimal ordering until the merged set contains $\geq t$ cells (line 9 Algorithm 1).

2. Similarly, merge P^s clusters in the end of the optimal ordering until the merged set contains $\geq t$ cells (line 11 Algorithm 1).

3. For every cluster P_i^s in the middle, determine the similarity between P_i^s and the cluster on the left and right, and merge P_i^s with the more similar cluster (line 20 - 26 Algorithm 1). Although the cluster on the left P_{i-1}^s always has $\geq n_{\min}^{\text{cluster}}$ cells, the cluster on the right P_{i+1}^s may have a minimum of one cell, so that the determined similarity between P_i^s and P_{i+1}^s is sensitive to noise. In order to improve the robustness of similarity comparison, instead of determining the similarity between P_i^s and P_{i+1}^s, we determine the similarity between P_i^s and $P_{\text{rmm}}^s = P_{i+1}^s \cup P_{i+2}^s \cup ... \cup P_{i+m}^s$ (line 20 Algorithm 1), where $|P_{\text{rmm}}^s| \geq n_{\min}^{\text{cluster}}$, $|P_{\text{rmm}}^s| - P_{i+m}^s < n_{\min}^{\text{cluster}}$, and rmm is the shorthand for right minimax.

4. Once $|P_i^s| \geq n_{\min}^{\text{cluster}}$, determine the similarity between P_{i-1}^s and P_i^s, and merge them if their similarity is above certain threshold, otherwise split them (line 32 - 36 Algorithm 1). The code length of merged $X_{\text{list}(P_{i-1}^s \cup P_i^s)}$ is

$$L_m = L(X_{\text{list}(P_{i-1}^s \cup P_i^s)}) \ .$$

The code length of divided $X_{\text{list}(P_{i-1}^s \cup P_i^s)}$ is

$$L_d = L(X_{\text{list}(P_{i-1}^s \cup P_i^s)}, \{P_{i-1}^s, P_i^s\}) \ .$$

If $L_d - L_m \geq -r_{\min}^{\text{mdl}} \text{abs}(L_m)$, merge P_{i-1}^s and P_i^s, otherwise split them. This conditional statement generalizes the situations where L_m is either non-negative or negative.

5. Continue inspecting the next cluster in P^s.

6. After inspecting all middle clusters, P^g is the final updated P^s.

We use a standard HAC tree to perform MIRAC when the number of samples is larger than ten thousand, which usually results in decreased but still reasonable performances (Figure S2 and Figure S3)). Although the optimal leaf ordering of the HAC tree is an important assumption, its computation takes too long when the number of samples is large. For example, it takes more than a week to compute the optimal leaf ordering of a dataset with about 68,000 samples.

2.3.2 Analysis of complexity

The time complexity of MIRAC is $O(n^4 + mn^2)$, where $n, m \geq 1$. The time complexity of computing the HAC tree with optimal leaf ordering is $O(n^4)$ (Bar-Joseph *et al.*, 2001). The time complexity of finding P^g is $O(mn^2)$, which is briefly analyzed as following.

The time complexity is $O(nm)$ for computing the code length $L(X_{n\times m}, P)$. In computing $L(X_{n\times m}, P)$, we compute the code lengths of each cluster and their labels. Since it takes $O(n)$ time to compute the code length of n observations with a single feature, computing the code length of all clusters and features individually takes $O(nm)$ time, and computing the code length of n labels take $O(n)$ time.

Similarly, the time complexity is $O(n_1 m + n_2 m)$ for computing the code length of $X_{n_1\times m}$ encoded by a model fitted with $X_{n_2\times m}$. Fitting a model on $X_{n_2\times m}$ takes $O(n_2 m)$ amount of time. Using the model to encode $X_{n_1\times m}$ takes $O(n_1 m)$ amount of time.

The time complexity of searching for P^g is $O(mn^2)$. In the worst case scenario, $|P^s| = n$, and the inspecting cluster is always merged with the left hand side cluster in step 4. However, the impact of $n_{\min}^{\text{cluster}}$ and step 3 on the time complexity is not straight forward, so we divide step 3 code length computation into three parts and analyze the upper bound of them individually. The divided three parts are the left cluster, inspecting cluster, and right minimax cluster. Because we keep increasing the size of the left cluster, the upper bound of computing the left cluster code length throughout the execution is $O(mn^2)$. The maximum size of the inspecting cluster is $n_{\min}^{\text{cluster}} - 1$, so the upper bound of computing the inspecting cluster code length is also $O(mn^2)$, when every middle singleton is evaluated $n_{\min}^{\text{cluster}} - 1$ times before merging with the left. The maximum size of the right minimax cluster is $2n_{\min}^{\text{cluster}} - 1$,so the upper bound of evaluating right minimax cluster is also $O(mn^2)$, when every increment of the left cluster takes $n_{\min}^{\text{cluster}}$ times of evaluating right minimax cluster. Summarizing these three parts, the overall upper bound is $O(mn^2)$.

The space complexity of MIRAC is $O(nm + n^2)$, which could be decomposed into the following three parts. The space complexity of storing the $n \times m$ data matrix X is $O(nm)$. The space complexity of storing the HAC tree with optimal leaf ordering is $O(n)$. The space complexity of storing the pairwise distance matrix is $O(n^2)$.

2.4 Cluster separating genes identification

We use XGBoost (Chen and Guestrin, 2016), a scalable and sparsity-aware tree boosting system, to identify genes that are able to separate a specific set of clusters. Although various statistical approaches have been developed to identify differentially expressed genes (Vallejos *et al.*, 2016, 2015; Kharchenko *et al.*, 2014; Qiu *et al.*, 2017; Korthauer *et al.*, 2016), we decided to use a machine learning based method due to feasibility and scalability.

Statistical approaches become less practical when the number of identified clusters becomes large. Since clusters could be compared in many ways, a large number of statistical tests could potentially be performed to identify differentially expressed genes. Each cluster C could be compared with any arbitrary subset of the other clusters. Biological interpretation could also be challenging, since a gene may be up-regulated, down-regulated, or non-significant in C when it is compared with different clusters. The scalability of the statistical approaches is also not well characterized for relatively large number of cells and clusters.

Rather than providing a meticulous p-value for each gene among the compared clusters, we rank the genes by their importances on separating the clusters under comparison. We use cross validation to train an XGBoost classifier on the compared clusters, and the classifier is essentially a bag of decision trees (Chen and Guestrin, 2016). The importance of a gene in the trained classifier is the number of times it has been used as a inner node of the decision trees. In order to alleviate the influences of stochasticity on interpretation, we use bootstrap with feature shuffling to better estimate the importances of genes in separating the compared clusters.

2.5 Quality control by k-nearest neighbor

We developed two straightforward methods based on k-nearest neighbor (KNN) to improve the signal to noise ratio in the data matrix. With relative large number of cells, we assume that any one of the cells is similar to at least k other cells in their transcriptomic profiles. With this assumption, we remove cell outliers and impute gene dropouts.

2.5.1 Detect cell outliers

We mark cells as outliers if they are distinct from their k-nearest neighbors, according to the pairwise similarity between cells. The Algorithm S1 could take multiple iterations, so that the most distinct cells could be marked at first and less distinct ones in the following iterations. Because outliers may have various biological and technical causes, samples and features in a dataset should only be removed after extensive exploratory data analysis and rigorous reasoning with domain specific knowledge.

2.5.2 Impute gene dropouts

In an scRNA-seq experiment, if a truly expressed gene is not detected in a cell, the gene is called being dropped out, and such events are called gene dropouts (Kharchenko *et al.*, 2014). Gene dropouts may be caused by biological and technical reasons (Kharchenko *et al.*, 2014; Risso *et al.*, 2018; Pierson and Yau, 2015; Zappia *et al.*, 2017). The rationale behind the possible causes of biological dropouts mainly involves transcriptional bursting (Suter *et al.*, 2011; Tantale *et al.*, 2016) and RNA degradation. With regard to technical dropouts, the main concerns are the relatively small number of RNA transcripts of a gene, amplification efficiency, and batch effect (Zappia *et al.*, 2017).

We exploit the transcriptomic profiles of the k-nearest neighbors of a cell to impute the gene dropouts in the cell (Algorithm S2)). The algorithm could take multiple iterations, so that the dropped-out genes that are expressed in all k-nearest neighbors could be imputed at first, and the ones that are expressed in most but not all of k-nearest neighbors could be imputed in the following iterations.

2.6 Benchmark

We benchmarked the clustering and quality control performances of `scedar` on simulated and experimental scRNA-seq datasets. We obtained previously published experimental scRNA-seq datasets (Table 1). We also generated 50 simulated scRNA-seq datasets with Splatter (Zappia *et al.*, 2017). The simulation parameters are estimated by Splatter according to a Drop-seq dataset (Macosko *et al.*, 2015) and a 10x Genomics GemCode dataset (Zheng *et al.*, 2017). Within each simulated dataset, the cells have 8 clusters taking approximately the following proportions: $\langle$ 0.3, 0.2, 0.15, 0.15, 0.05, 0.05, 0.05, 0.05 $\rangle$, with a gene dropout rate around 5%.

We performed all benchmark analyses on a high-performance computing cluster, of which the computing resources are strictly managed by Univa Grid Engine. The cluster nodes have CPUs of Intel Xeon E5-2680 v3 or Intel Xeon E7-8880 v3. The memory sizes are either 128GB, 256GB, or 1TB. When scheduling analytical jobs for benchmarking, we make sure that the number of cores and allocated memory are enough for the program.

2.6.1 Clustering

The clustering accuracy and stability of MIRAC were benchmarked together with several other clustering methods on experimental scRNA-seq datasets listed in Table 1.

The compared clustering methods are:

- MIRAC on 2D t-SNE projection.
- single-cell consensus clustering (SC3) (Kiselev *et al.*, 2017).

- K-means clustering on 2D t-SNE projection.
- Hierarchical agglomerative clustering on 2D t-SNE projection.
- Density-based spatial clustering of applications with noise (DBSCAN) (Ester *et al.*, 1996) on 2D t-SNE projection.

Although MIRAC could be directly applied on the expression matrix, dimensionality reduction is able to improve the clustering performance when the number of features is high (Aggarwal *et al.*, 2001). Among many dimensionality reduction methods, we chose t-SNE in order to visualize the clustering results more clearly.

When benchmarking for accuracy, we cluster the cells in each experimental dataset using the compared clustering methods with a grid of parameters. The maximum similarities between the clustering results and the cell types from the publications are used to compare the accuracy of different clustering methods. Although taking the maximum increases the chance of overfitting, it resembles the procedure of clustering analysis in practice.

Similarly, when benchmarking for stability, we cluster the cells in each experimental dataset using the compared methods with the same parameters but ten different random states. The similarity of clustering results between different random states are used to compare the stability of different clustering methods.

We use two cluster similarity metrics, cluster consistent ratio (CCR) and adjusted Rand index (ARI) (Hubert and Arabie, 1985) for measuring the accuracy and stability of clustering methods respectively. When we have a coarse reference partition P_r and a finer clustering partition P_c, the CCR is computed as the ratio of pairs within each cluster of P_c that are also in the same cluster of P_r, with the number of clusters kept the same across compared methods. The ARI is computed with the Python package `scikit-learn` (Pedregosa *et al.*, 2011) using the mathematical formula in Hubert and Arabie (1985).

We used CCR to measure clustering accuracy rather than ARI, because it greatly penalizes the split of a large cluster in P_r into multiple smaller ones in P_c. In contrast, CCR does not penalize such clustering behavior, in order to support the evaluation of the transcriptomic variety within a well known cell type, which is an important goal of scRNA-seq experiments that cannot be easily achieved by bulk RNA-seq experiments. In addition, we also provide a method in `scedar` to easily merge multiple clusters together, in case the users found the sub-types of a cell type are very similar to each other.

However, we used ARI to measure clustering stability rather than CCR, because the differences between P_r and P_c are completely caused by different random states, hence splitting a cluster in P_r should be penalized.

2.6.2 Cell outlier detection

We visualize real datasets before and after removing cell outliers in t-SNE projection and pairwise distance heatmaps, without quantitative evaluations like receiver operating characteristic (ROC) curve, since outliers are not well defined in high-dimensional space (Aggarwal and Yu, 2001; Aggarwal *et al.*, 2001). Approaches to identifying outliers in low-dimensional spaces (≤ 3) does not scale well to higher dimensions due to the exponentially decreased density of data points in the space and increased instability of distances.

2.6.3 Gene dropout imputation

We simulate gene dropouts using Splatter (Zappia *et al.*, 2017) to obtain a dropout rate around 5%. Then, we benchmark the performance of imputing gene dropout as two parts, detection and smoothing. On simulated data, because the true gene dropouts are known, we use ROC curve and mean squared errors (MSEs) to characterize the performance of gene dropout detection and smoothing respectively. On real data, we visualize the cells in 2D t-SNE space before and after imputation. The compared methods are KNN gene dropout imputation (KNNGDI), SAVER (Huang *et al.*, 2018), MAGIC (van Dijk *et al.*, 2018), and scImpute (Li and Li, 2018).

2.7 Data availability

The real datasets can be obtained from `https://hemberg-lab.github.io/scRNA.seq.datasets/` and `https://support.10xgenomics.com/single-cell-gene-expression/datasets`. The simulated datasets can be generated using scripts at `https://github.com/logstar/scedar/tree/master/docs/r_scripts`.

3 Results

3.1 Basic workflow of `scedar`

We illustrate the basic workflow of using `scedar` for scRNA-seq data analysis with the dataset published by Zeisel *et al.* (2015) (Figure 1). The dataset contains the RNA unique molecule identifier (UMI) counts of 19,972 genes in 3005 cells from mouse brain. We selected this dataset for demonstration because it could be clearly visualized in small graphs, although our package is capable of analyzing much larger datasets (Figure S2 and S3)).

In Figure 1, each box represents a data analysis step, and they are consecutively executed according to the arrow. The purpose and runtime are listed in the upper ribbon. The code that are essential to the step is listed in the box, and their results are also shown.

The preparation step imports required packages and loads the data. The class SampleDistanceMatrix is one of the core data structures in the package that is used to store the read counts and pairwise distances of an scRNA-seq dataset. Because the pairwise distance computation is delayed until necessary, i.e. lazily loaded, this step only takes 12 seconds. We use cosine distance rather than correlation distance to greatly speed up the computation, since we implemented the computation procedure of pairwise cosine distances completely with numpy linear algebra operations implemented with OpenBLAS in the C programming language.

The t-SNE scatter plot and KNN graph are used to explore the dataset. The cell type labels published by Zeisel *et al.* (2015) are truncated to fit in the space. We also provide methods to visualize arbitrary statistics, e.g. number of expressed genes, of individual cells as color gradient. The layouts of cells in t-SNE and KNN graph are similar to each other. Although the runtime for KNN graph is much shorter, optimizing the parameter or computation procedure (Linderman *et al.*, 2017) for t-SNE could greatly speed up the computation.

The MIRAC step clusters the cells and visualizes them with t-SNE scatter plot and pairwise distance matrix heatmap. The heatmap generation procedure in `scedar` is optimized for large-scale datasets, which is able to generate a heatmap with tens of thousands of columns and rows in a few minutes. Users could also generate heatmaps for the read count matrix to directly inspect the sparsity of datasets (Figure S2 B and S3 B).

The last step identifies cluster separating genes with XGBoost (Chen and Guestrin, 2016). Users could choose an arbitrary set of clusters to compare, and the genes are each ranked by their

importance in separating the clusters. Then, the read counts of a gene across clustered labels could easily be visualized by t-SNE scatter plot.

3.2 Performance of MIRAC clustering

We benchmarked several clustering methods on the datasets listed in Table 1 (Figure 2). Each dataset is clustered multiple times with each clustering method to obtain different numbers of clusters (Table S1).

The t-SNE based clustering methods are faster than SC3 (Figure 2A). Also, the t-SNE based clustering methods have similar runtimes with their lines overlapping in the Figure 2A, since the time limiting step is the computation of t-SNE projection.

The CCRs of t-SNE based clustering methods are comparable to SC3. The CCRs of Zheng *et al.* (2017) and Macosko *et al.* (2015) datasets are not included in Figure 2 B, since the SC3 cluster labels are `NA`s in all 10 runs with different number of clusters. However, the representative MIRAC clustering results of Zheng *et al.* (2017) and Macosko *et al.* (2015) datasets are visualized with t-SNE scatter plot and pairwise distance matrix heatmap (Figure S2 and S3). For smaller datasets, the representative MIRAC results are shown in Figure S4 . Although t-SNE projections obtained with different random states are distinct from each other, the consistency of clustering results is comparable to SC3 (Figure S1).

3.3 Performance of imputing gene dropouts

We benchmarked several gene dropout imputation methods on the simulated 10x Genomics (Figure 3) and Drop-seq (Figure S5) datasets.

KNNGDI is faster than other compared methods (Figure 3A). MAGIC runtime grows slowly with regard to the number cells because the time limiting step is performed on dimensionality reduced data.

The performance of KNNGDI on detecting gene dropouts is comparable to SAVER and better than scImpute and MAGIC. The ROC curve of scImpute sharply turns around TPR = 0.25 (Figure 3B and S5B) because its threshold parameters, determining whether a zero entry is a dropout or not, are not sensitive enough to achieve any higher TPRs. Although the AUCs of KNNGDI and SAVER are higher than scImpute and MAGIC, they all have comparable performances when FPRs are lower than 0.05 (Figure 3B), except that MAGIC has worse performance on the simulated Drop-seq datasets with higher sparsity in the Zeisel *et al.* (2015) dataset (Figure S5B).

The MSEs of KNNGDI on correcting gene dropouts are comparable to SAVER and smaller than scImpute and MAGIC (Figure S5 C and S5 D). However, these methods all have MSEs many times higher than the MSEs of the observed counts to the true counts, which implies that these methods all introduced many times more reads than the true dropout reads.

None of the compared gene dropout imputation methods greatly improved the separation of different cell types (Figure 3C, S2 E, and S3 E) in the t-SNE projection of the large experimental datasets. On the other hand, the small datasets with higher depth do not need gene dropout imputations for cell type separation (Figure S4)). Notably, MAGIC incorrectly mixed mural and endothelial cells in the Zeisel *et al.* (2015) dataset (Figure 3C).

3.4 Performance of detecting cell outliers

We detected and removed cell outliers in datasets listed in Table 1 with the KNN method (Figure 4, S4 , S2 , and S3). Although the detection method has many limitations, outliers tend to be points on the t-SNE scatter plots either far away from the majority of their same types or along the

edges of a group of agglomerated points. On the pairwise distance matrix heatmap, outliers tend to be small chunks that are distinct from their neighbors, and the heatmap becomes smoother after removing the outliers.

3.5 Identification of cluster separating genes

We used `scedar` to identify the genes distinguishing the MIRAC cluster 1, 15, and 22 of the Zeisel *et al.* (2015) dataset (Figure 5 and Table S2)). In the original publication, the upper to lower MIRAC cluster 22, 1, and 15 in the t-SNE scatter plot are assigned to microglia, endothelial cells, and astrocytes respectively (Figure 3C). We choose these three clusters to inspect one of the discrepancies between cell types and MIRAC clustering results, where the small isolated upper part of the MIRAC cluster 15 is assigned to microglia instead of endothelial cells in the original publication.

The smaller upper isolated part of MIRAC cluster 15 might be a distinct cell sub-type of microglia. Although it expresses a microglia marker gene C1qb (Figure 5B) (Beutner *et al.*, 2013), it does not express Mrc1 or Apoe in the same pattern as the MIRAC cluster 22 (Figure 5B and 5C). According the transcription levels of the cluster separating genes (Figure 5C), the smaller upper isolated part of MIRAC cluster 15, which is located at the top of the cluster 15 rows, has some genes expressed in the same pattern as the microglia, but it also has some other genes expressed distinctly from the microglia.

4 Discussion

Comprehensive profiling of the transcriptomes of individual cells within an organism or tissue by scRNA-seq is able to facilitate the systematic study of physiological or pathological states (Regev *et al.*, 2017; Baslan and Hicks, 2017; Cao *et al.*, 2017). Previous scRNA-seq experiments identified novel cell types or states (Villani *et al.*, 2017), obtained insights into the regulation of differentiation process (Cao *et al.*, 2017; Rizvi *et al.*, 2017), and inferred molecular mechanisms of tissue functions (Zeisel *et al.*, 2015; Haber *et al.*, 2017; Lake *et al.*, 2018).

The biological results of scRNA-seq experiments are obtained from extensive data analyses, which could take more time than doing the experiments. As the size of scRNA-seq datasets increases rapidly (Svensson *et al.*, 2018; Rozenblatt-Rosen *et al.*, 2017), computational methods also need to be scalable by exploiting hardware and software techniques for big data analytics, in order to complete an analysis within a reasonable amount of time.

`Scedar` is able to facilitate quality control, clustering, and identification of cluster separating genes for scRNA-seq data by exploiting scalable system design patterns. We parallelized time-consuming computational procedures by multi-processing without excessive copies of shared data in memory. We also decompose the whole analytical procedure into multiple steps, so that certain steps could be specifically optimized without repeating others. In addition, intermediate results, such as pairwise distances and t-SNE projections, are lazy loaded and cached in a unified data structure to speed up analytical routines, prevent repeated computations, and alleviate the burden on users to keep track of all intermediate results.

In `scedar`, we developed a clustering algorithm, MIRAC, for scRNA-seq data. MIRAC clusters observations in three steps: 1) build a tree by hierarchical clustering, 2) divide the tree into sub-trees, and 3) merge similar sub-trees into individual clusters. This clustering strategy adapts the basic ideas of BIRCH (Zhang *et al.*, 1996) and BackSPIN (Zeisel *et al.*, 2015). Instead of building a balanced tree structure, like the clustering feature tree in BIRCH for further partitioning, MIRAC divides the tree structure built by hierarchical clustering optionally in a balanced manner (Figure S6

and Supplementary Method section 1.3), which simplifies the clustering procedure. In BackSPIN, a sorted pairwise correlation matrix is recursively bi-partitioned into clusters according a criteria based on the normalized sums of correlation coefficients. However, MIRAC iteratively merges the relatively small sub-clusters according a criteria based on the MDL principle, which increases the robustness for determining whether two groups of observations should be put in the same cluster or not. Especially, when the number of observations is large within a group, finding an optimal bi-partition is not straightforward, since there may be multiple distinct sub-groups.

There are still many possible improvements on `scedar`. To make the clustering analysis more flexible, we could implement a user interface for supervised clustering by selecting points on two or three dimensional projections. To make MIRAC results more intuitive, we could visualize the clustering process on the hierarchical tree, which could also facilitate parameter optimization. To improve the scalability of MIRAC, we could provide more efficient methods to obtain the optimal leaf ordering using linear embedding techniques (Aydin *et al.*, 2016). To improve the scalability of the backend data structure, we could extend it with distributed analytic systems such as Apache Spark (Zaharia *et al.*, 2016). To improve the scalability of KNN methods, we could also provide implementations with approximate nearest neighbor methods (Andoni and Indyk, 2006; Kushilevitz *et al.*, 2000).

5 Conclusion

`Scedar` is an efficient and extensible Python-based platform for large scale scRNA-seq exploratory data analysis, which will facilitate biological interpretations in specific domains. The package provides a straightforward user interface to perform quality control, clustering analysis, and visualization. In addition, the modular design and comprehensive tests of `scedar` facilitate customization for specific use cases, extensibility for incorporating analytical procedures, and for future refactoring to further improve scalability.

6 Acknowledgements

We thank Anthony Cros for helpful discussions on the software design of `scedar`. We thank Jim Zhe Zhang for helpful discussions on the clustering performances.

7 Funding

This research is supported by the Department of Biomedical and Health Informatics at the Children's Hospital of Philadelphia.

8 Algorithms

input : X, a numeric data matrix of size $n \times m$
$n_{\text{min}}^{\text{cluster}}$, minimum number of rows for a cluster
$r_{\text{min}}^{\text{mdl}}$, minimum ratio of MDL reduction for increasing the number of cluster
output: Clusters of n rows a list of n $\langle$row index, cluster index$\rangle$ pairs

```
// Phase 1:  divide n rows into sub-clusters
// build HAC tree with linear optimal leaf ordering
hacTree ← HAClinearOptLeafOrd(X)
// divide the tree leaves into sub-clusters with less than n_min^cluster rows
subClusters ← DivideHACTree (hacTree, n_min^cluster)
// Phase 2:  iteratively merge sub-clusters into clusters
// Merge beginning and end sub-clusters to form two minimum sub-clusters
// with more than n_min^cluster rows
MergeSubCls (subClusters, start = 1, minSize =n_min^cluster)
n_sub-clusters ← Length (subClusters)
MergeSubCls (subClusters, start =n_sub-clusters, step = −1, minSize =n_min^cluster)
i ← 2
while i ≤ n_sub-clusters do
    left ← subClusters [i − 1]
    this ← subClusters [i]
    // merge this with left or right until this has ≥ n_min^cluster rows
    while Length (this) < n_min^cluster and i + 1 ≤ n_sub-clusters do
        right ← subClusters [i + 1]
        // minimum merged sub-cluster with ≥ n_min^cluster rows starting from i + 1
        rightMM ← MergeSubCls (subClusters, start = i + 1, minSize =n_min^cluster, inPlace =false)
        if EncodeMDL (left, this) < EncodeMDL (rightMM, this) then // merge left and this
            left ← MergeSubCls (subClusters, selected = [i − 1, i])
            this ← right
        else // merge this and right
            this ← MergeSubCls (subClusters, selected = [i, i + 1])
        end
    end
    // decide whether merge left and this or not
    merged ← Concatenate (left, this)
    mergedMDL ← MDL (merged)
    splitMDL ← MDL (left) + MDL (this) + SplitOverheadMDL (left, this)
    if splitMDL < MaxSplitMDL (mergedMDL, r_min^mdl) then // split
        i ← i + 1
    else // merge
        MergeSubCls (subClusters, selected = [i − 1, i])
    end
end
return subClusters
```

Algorithm 1: Minimum description length iteratively regulated agglomerative clustering

9 Figures

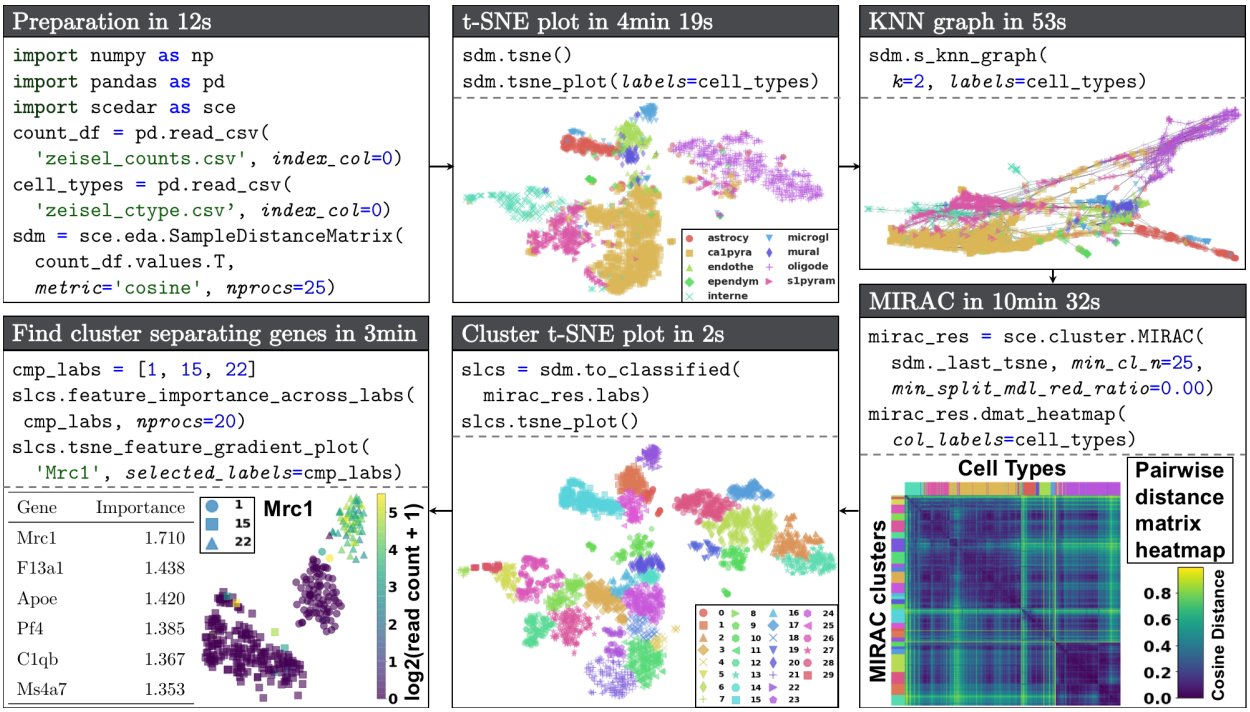

Figure 1: Demo of `scedar`. Workflow of using `scedar` to analyze an scRNA-seq dataset with 3005 mouse brain cells and 19,972 genes generated using the STRT-Seq UMI protocol by Zeisel *et al.* (2015). Procedures and parameters that are not directly related to data analysis are omitted. The full version of the demo is available at `https://github.com/logstar/scedar/tree/master/docs/notebooks`.

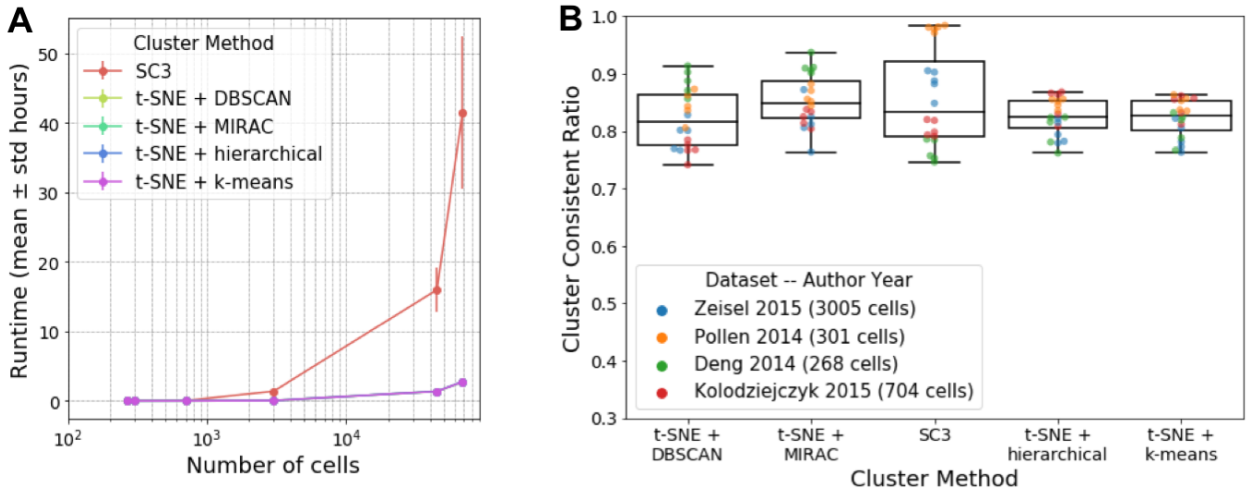

Figure 2: Clustering method benchmarks on experimental datasets. A) Runtimes. B) CCRs on different datasets, with different points of each dataset representing different numbers of clusters. For each dataset, the numbers of clusters are the same across all compared clustering methods.

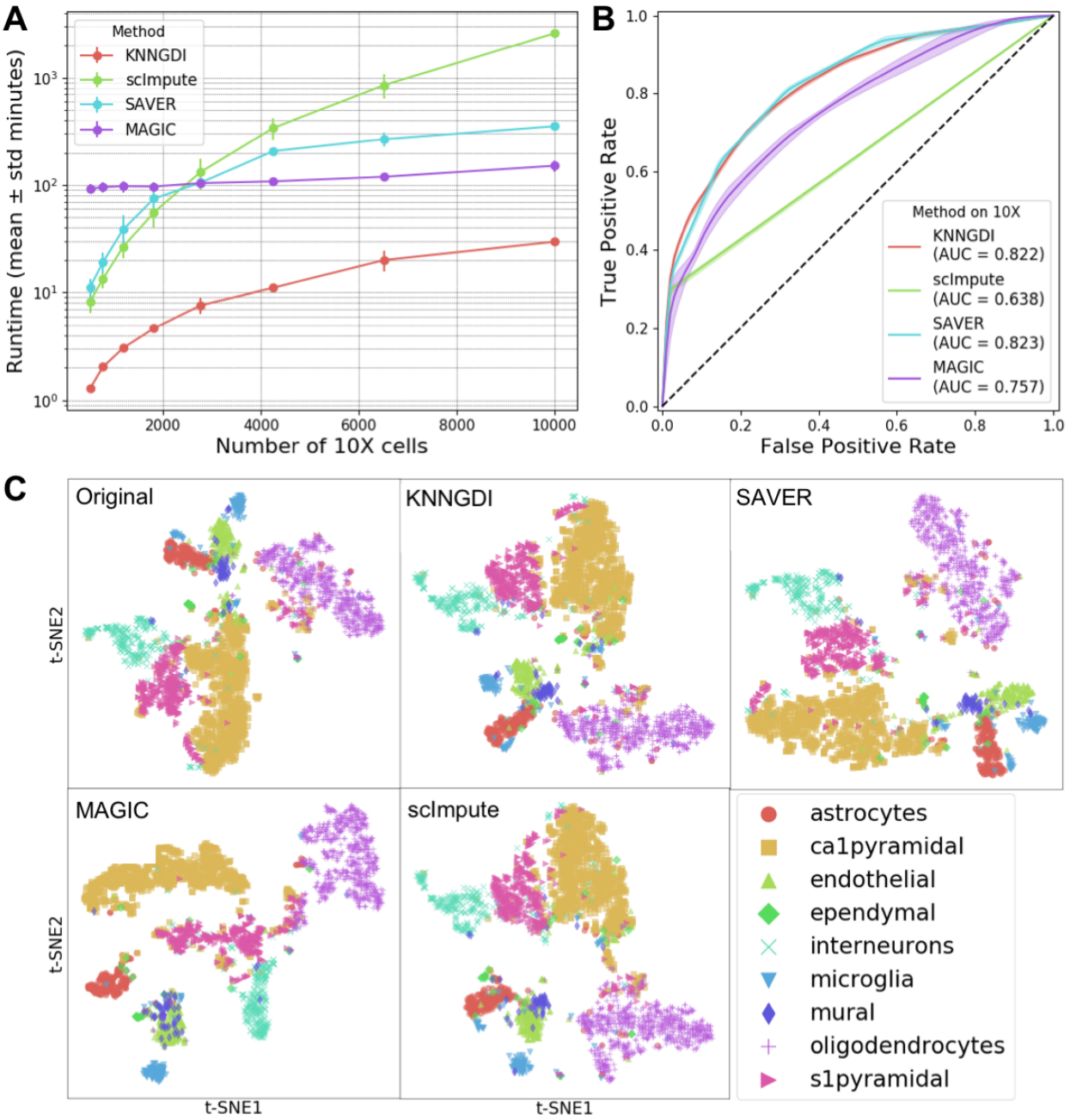

Figure 3: Gene dropout imputation method benchmarks. A) Runtimes on 40 simulated 10x Genomics datasets. B) ROC curves (± standard deviation) of dropout detection on the simulated 10x Genomics datasets. C) t-SNE scatter plots of the Zeisel *et al.* (2015) dataset after gene dropout imputations.

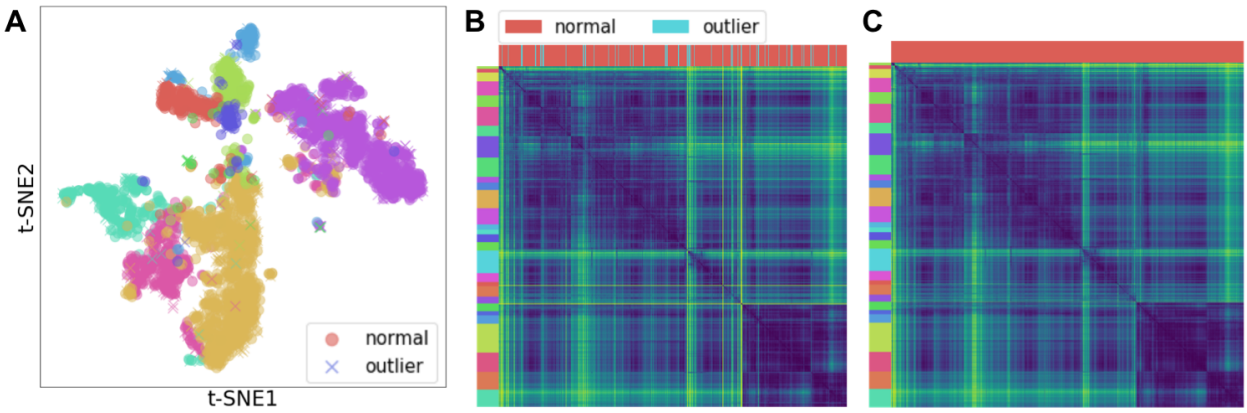

Figure 4: KNN cell outlier detection on the Zeisel *et al.* (2015) dataset. A) t-SNE scatter plot with color as cell type labels and marker shape as normal or outlier labels. 9.3% cells are marked as outlier. B) Pairwise cosine distance heatmap with left strip as MIRAC labels and upper strip as normal or outlier labels. C) Pairwise cosine distance heatmap with outliers removed.

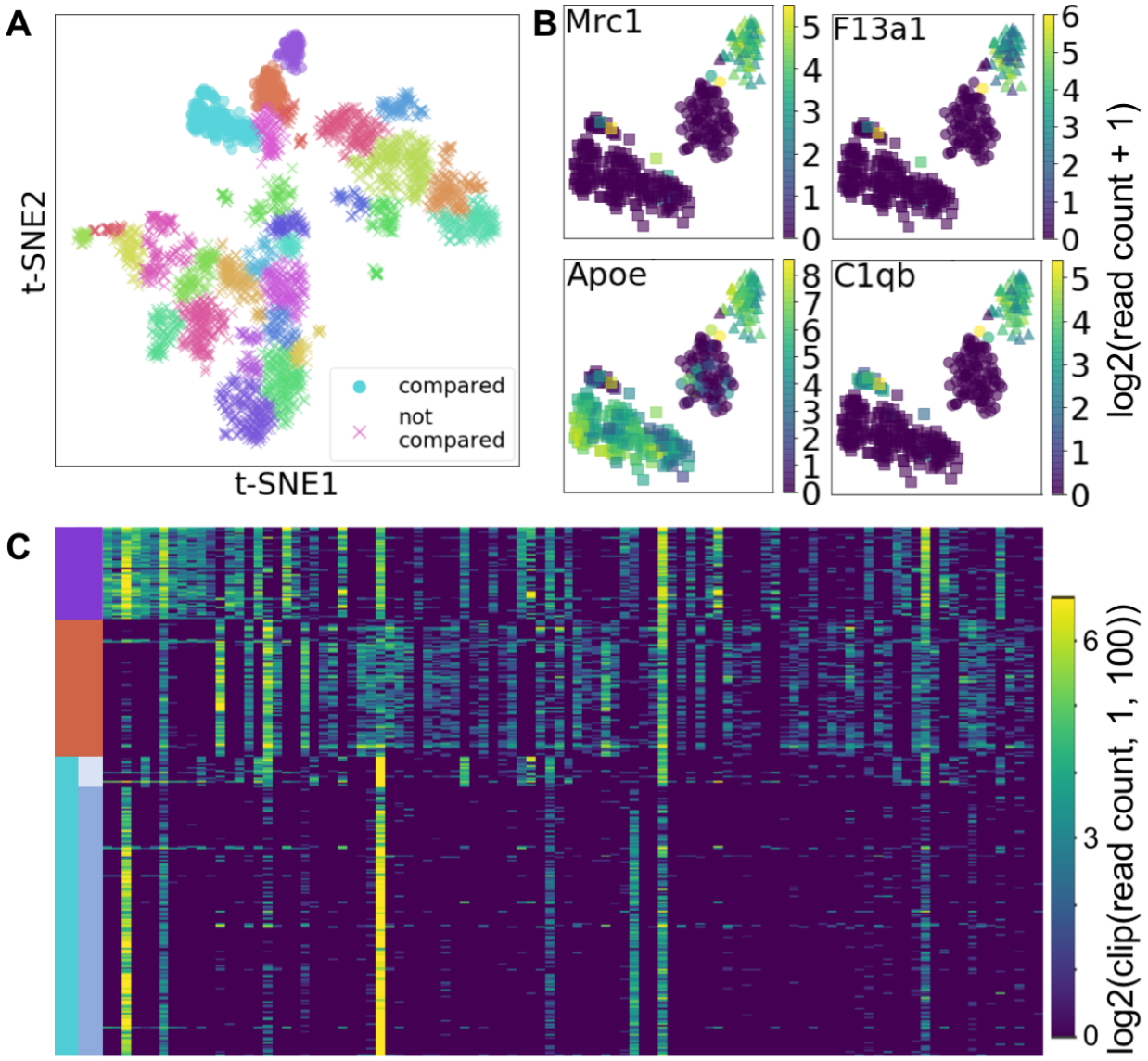

Figure 5: Identified genes separating the MIRAC clusters 1, 15, and 22 of the Zeisel *et al.* (2015) dataset. A) t-SNE scatter plot with color as MIRAC cluster labels and marker shape as compared or not compared. B) t-SNE scatter plots of the compared clusters with color as $\log_2(\text{read count} + 1)$ of the corresponding gene and marker shape as MIRAC clusters. C) Transcription level heatmap of the top 100 important cluster separating genes in the compared cells, with rows as cells ordered by cluster labels and columns as genes ordered by importances. The color gradient is $\log_2(\text{clip}(\text{read count}, 1, 100))$, where the $\text{clip}(\text{read count}, 1, 100)$ function changes any read count below 1 to 1 and above 100 to 100, in order to better compare genes at different transcription levels.

Table 1: Real scRNA-seq datasets for benchmark

Publication	Organism	Tissue	Protocol	# cells	# genes
Macosko *et al.* (2015)	Mus musculus	Retina	Drop-seq	44,808	23,288
Deng *et al.* (2014)	Mus musculus	Embryo	Smart-Seq / Smart-Seq2	268	22,431
Kolodziejczyk *et al.* (2015)	Mus musculus	Embryonic stem cell	SMARTer	704	38,653
Pollen *et al.* (2014)	Homo sapiens	dermal, blood, pluripotent, and neural	SMARTer	301	23,730
Zeisel *et al.* (2015)	Mus musculus	brain	STRT-Seq UMI	3005	19,972
Zheng *et al.* (2017)	Homo sapiens	peripheral blood mononuclear cell	GemCode	68,579	33,694

Source: `https://hemberg-lab.github.io/scRNA.seq.datasets/`

References

Abramson, M. (1976). Restricted combinations and compositions. *Fibonacci Quart*, **14**(5), 439.

Aggarwal, C. C. and Yu, P. S. (2001). Outlier Detection for High Dimensional Data. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, SIGMOD '01, pages 37–46, New York, NY, USA. ACM.

Aggarwal, C. C., Hinneburg, A., and Keim, D. A. (2001). On the Surprising Behavior of Distance Metrics in High Dimensional Space. In G. Goos, J. Hartmanis, J. van Leeuwen, J. Van den Bussche, and V. Vianu, editors, *Database Theory — ICDT 2001*, volume 1973, pages 420–434. Springer Berlin Heidelberg, Berlin, Heidelberg.

Aibar, S., González-Blas, C. B., Moerman, T., Huynh-Thu, V. A., Imrichova, H., Hulselmans, G., Rambow, F., Marine, J.-C., Geurts, P., Aerts, J., van den Oord, J., Atak, Z. K., Wouters, J., and Aerts, S. (2017). SCENIC: Single-cell regulatory network inference and clustering. *Nat. Methods*, **14**(11), 1083–1086.

Andoni, A. and Indyk, P. (2006). Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 459–468.

Aydin, K., Bateni, M., and Mirrokni, V. (2016). Distributed Balanced Partitioning via Linear Embedding. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, WSDM '16, pages 387–396, New York, NY, USA. ACM.

Bar-Joseph, Z., Gifford, D. K., and Jaakkola, T. S. (2001). Fast optimal leaf ordering for hierarchical clustering. *Bioinformatics*, **17**(suppl_1), S22–S29.

Baslan, T. and Hicks, J. (2017). Unravelling biology and shifting paradigms in cancer with single-cell sequencing. *Nature Reviews Cancer*, **17**(9), 557–569.

Beutner, C., Linnartz-Gerlach, B., Schmidt, S. V., Beyer, M., Mallmann, M. R., Staratschek-Jox, A., Schultze, J. L., and Neumann, H. (2013). Unique transcriptome signature of mouse microglia: Transcriptome Analysis of Microglia. *Glia*, **61**(9), 1429–1442.

Cao, J., Packer, J. S., Ramani, V., Cusanovich, D. A., Huynh, C., Daza, R., Qiu, X., Lee, C., Furlan, S. N., Steemers, F. J., Adey, A., Waterston, R. H., Trapnell, C., and Shendure, J. (2017). Comprehensive single-cell transcriptional profiling of a multicellular organism. *Science*, **357**(6352), 661–667.

Chen, T. and Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *arXiv:1603.02754 [cs]*, pages 785–794.

Deng, Q., Ramsköld, D., Reinius, B., and Sandberg, R. (2014). Single-Cell RNA-Seq Reveals Dynamic, Random Monoallelic Gene Expression in Mammalian Cells. *Science*, **343**(6167), 193–196.

Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A Density-based Algorithm for Discovering Clusters a Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231, Portland, Oregon. AAAI Press.

Haber, A. L., Biton, M., Rogel, N., Herbst, R. H., Shekhar, K., Smillie, C., Burgin, G., Delorey, T. M., Howitt, M. R., Katz, Y., Tirosh, I., Beyaz, S., Dionne, D., Zhang, M., Raychowdhury, R., Garrett, W. S., Rozenblatt-Rosen, O., Shi, H. N., Yilmaz, O., Xavier, R. J., and Regev, A. (2017). A single-cell survey of the small intestinal epithelium. *Nature*, **551**(7680), 333–339.

Hagberg, A. A., Schult, D. A., and Swart, P. J. (2008). Exploring Network Structure, Dynamics, and Function using NetworkX. page 5.

Hansen, M. H. and Yu, B. (2001). Model Selection and the Principle of Minimum Description Length. *Journal of the American Statistical Association*, **96**(454), 746–774.

Harrow, J., Frankish, A., Gonzalez, J. M., Tapanari, E., Diekhans, M., Kokocinski, F., Aken, B. L., Barrell, D., Zadissa, A., Searle, S., Barnes, I., Bignell, A., Boychenko, V., Hunt, T., Kay, M., Mukherjee, G., Rajan, J., Despacio-Reyes, G., Saunders, G., Steward, C., Harte, R., Lin, M., Howald, C., Tanzer, A., Derrien, T., Chrast, J., Walters, N., Balasubramanian, S., Pei, B., Tress, M., Rodriguez, J. M., Ezkurdia, I., van Baren, J., Brent, M., Haussler, D., Kellis, M., Valencia, A., Reymond, A., Gerstein, M., Guigo, R., and Hubbard, T. J. (2012). GENCODE: The reference human genome annotation for The ENCODE Project. *Genome Research*, **22**(9), 1760–1774.

Huang, M., Wang, J., Torre, E., Dueck, H., Shaffer, S., Bonasio, R., Murray, J. I., Raj, A., Li, M., and Zhang, N. R. (2018). SAVER: Gene expression recovery for single-cell RNA sequencing. *Nature Methods*, page 1.

Hubert, L. and Arabie, P. (1985). Comparing partitions. *Journal of Classification*, **2**(1), 193–218.

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science Engineering*, **9**(3), 90–95.

Jacomy, M., Venturini, T., Heymann, S., and Bastian, M. (2014). ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software. *PLOS ONE*, **9**(6), e98679.

Kharchenko, P. V., Silberstein, L., and Scadden, D. T. (2014). Bayesian approach to single-cell differential expression analysis. *Nature Methods*, **11**(7), 740–742.

Kiselev, V. Y., Kirschner, K., Schaub, M. T., Andrews, T., Yiu, A., Chandra, T., Natarajan, K. N., Reik, W., Barahona, M., Green, A. R., and Hemberg, M. (2017). SC3: Consensus clustering of single-cell RNA-seq data. *Nature Methods*, **14**(5), 483–486.

Kolodziejczyk, A. A., Kim, J. K., Tsang, J. C. H., Ilicic, T., Henriksson, J., Natarajan, K. N., Tuck, A. C., Gao, X., Bühler, M., Liu, P., Marioni, J. C., and Teichmann, S. A. (2015). Single Cell RNA-Sequencing of Pluripotent States Unlocks Modular Transcriptional Variation. *Cell Stem Cell*, **17**(4), 471–485.

Korthauer, K. D., Chu, L.-F., Newton, M. A., Li, Y., Thomson, J., Stewart, R., and Kendziorski, C. (2016). A statistical approach for identifying differential distributions in single-cell RNA-seq experiments. *Genome Biology*, **17**, 222.

Kushilevitz, E., Ostrovsky, R., and Rabani, Y. (2000). Efficient Search for Approximate Nearest Neighbor in High Dimensional Spaces. *SIAM Journal on Computing*, **30**(2), 457–474.

Lake, B. B., Chen, S., Sos, B. C., Fan, J., Kaeser, G. E., Yung, Y. C., Duong, T. E., Gao, D., Chun, J., Kharchenko, P. V., and Zhang, K. (2018). Integrative single-cell analysis of transcriptional and epigenetic states in the human adult brain. *Nature Biotechnology*, **36**(1), 70–80.

Li, W. V. and Li, J. J. (2018). An accurate and robust imputation method scImpute for single-cell RNA-seq data. *Nature Communications*, **9**(1).

Linderman, G. C., Rachh, M., Hoskins, J. G., Steinerberger, S., and Kluger, Y. (2017). Efficient Algorithms for t-distributed Stochastic Neighborhood Embedding. *arXiv:1712.09005 [cs, stat]*.

Macosko, E. Z., Basu, A., Satija, R., Nemesh, J., Shekhar, K., Goldman, M., Tirosh, I., Bialas, A. R., Kamitaki, N., Martersteck, E. M., Trombetta, J. J., Weitz, D. A., Sanes, J. R., Shalek, A. K., Regev, A., and McCarroll, S. A. (2015). Highly Parallel Genome-wide Expression Profiling of Individual Cells Using Nanoliter Droplets. *Cell*, **161**(5), 1202–1214.

McInnes, L. and Healy, J. (2018). UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv:1802.03426 [cs, stat]*.

McKinney, W. (2010). Data Structures for Statistical Computing in Python. pages 51–56.

Müllner, D. (2011). Modern hierarchical, agglomerative clustering algorithms. *arXiv:1109.2378 [cs, stat]*.

Pearson, K. (1901). LIII. *On lines and planes of closest fit to systems of points in space*. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, **2**(11), 559–572.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, **12**(Oct), 2825–2830.

Pierson, E. and Yau, C. (2015). ZIFA: Dimensionality reduction for zero-inflated single-cell gene expression analysis. *Genome Biology*, **16**(1).

Pollen, A. A., Nowakowski, T. J., Shuga, J., Wang, X., Leyrat, A. A., Lui, J. H., Li, N., Szpankowski, L., Fowler, B., Chen, P., Ramalingam, N., Sun, G., Thu, M., Norris, M., Lebofsky, R., Toppani, D., Kemp, 2nd, D. W., Wong, M., Clerkson, B., Jones, B. N., Wu, S., Knutsson, L., Alvarado, B., Wang, J., Weaver, L. S., May, A. P., Jones, R. C., Unger, M. A., Kriegstein, A. R., and West, J. A. A. (2014). Low-coverage single-cell mRNA sequencing reveals cellular heterogeneity and activated signaling pathways in developing cerebral cortex. *Nat. Biotechnol.*, **32**(10), 1053–1058.

Qiu, X., Hill, A., Packer, J., Lin, D., Ma, Y.-A., and Trapnell, C. (2017). Single-cell mRNA quantification and differential analysis with Census. *Nature Methods*, **14**(3), 309–315.

Regev, A., Teichmann, S. A., Lander, E. S., Amit, I., Benoist, C., Birney, E., Bodenmiller, B., Campbell, P., Carninci, P., Clatworthy, M., Clevers, H., Deplancke, B., Dunham, I., Eberwine, J., Eils, R., Enard, W., Farmer, A., Fugger, L., Göttgens, B., Hacohen, N., Haniffa, M., Hemberg, M., Kim, S., Klenerman, P., Kriegstein, A., Lein, E., Linnarsson, S., Lundberg, E., Lundeberg, J., Majumder, P., Marioni, J. C., Merad, M., Mhlanga, M., Nawijn, M., Netea, M., Nolan, G., Pe'er, D., Phillipakis, A., Ponting, C. P., Quake, S., Reik, W., Rozenblatt-Rosen, O., Sanes, J., Satija, R., Schumacher, T. N., Shalek, A., Shapiro, E., Sharma, P., Shin, J. W., Stegle, O., Stratton, M., Stubbington, M. J. T., Theis, F. J., Uhlen, M., van Oudenaarden, A., Wagner, A., Watt,

F., Weissman, J., Wold, B., Xavier, R., Yosef, N., and Participants, H. C. A. M. (2017). Science Forum: The Human Cell Atlas. *eLife*, **6**, e27041.

Risso, D., Perraudeau, F., Gribkova, S., Dudoit, S., and Vert, J.-P. (2018). A general and flexible method for signal extraction from single-cell RNA-seq data. *Nature Communications*, **9**(1).

Rizvi, A. H., Camara, P. G., Kandror, E. K., Roberts, T. J., Schieren, I., Maniatis, T., and Rabadan, R. (2017). Single-cell topological RNA-seq analysis reveals insights into cellular differentiation and development. *Nature Biotechnology*, **35**(6), 551–560.

Rozenblatt-Rosen, O., Stubbington, M. J. T., Regev, A., and Teichmann, S. A. (2017). The Human Cell Atlas: From vision to reality. *Nature News*, **550**(7677), 451.

Suter, D. M., Molina, N., Gatfield, D., Schneider, K., Schibler, U., and Naef, F. (2011). Mammalian Genes Are Transcribed with Widely Different Bursting Kinetics. *Science*, **332**(6028), 472–474.

Svensson, V., Vento-Tormo, R., and Teichmann, S. A. (2018). Exponential scaling of single-cell RNA-seq in the past decade. *Nature Protocols*, **13**(4), 599–604.

Tantale, K., Mueller, F., Kozulic-Pirher, A., Lesne, A., Victor, J.-M., Robert, M.-C., Capozi, S., Chouaib, R., Bäcker, V., Mateos-Langerak, J., Darzacq, X., Zimmer, C., Basyuk, E., and Bertrand, E. (2016). A single-molecule view of transcription reveals convoys of RNA polymerases and multi-scale bursting. *Nature Communications*, **7**, 12248.

Travis E. Oliphant (2006). *Guide to Numpy*.

žurauskienė, J. and Yau, C. (2016). pcaReduce: Hierarchical clustering of single cell transcriptional profiles. *BMC Bioinformatics*, **17**, 140.

Vallejos, C. A., Marioni, J. C., and Richardson, S. (2015). BASiCS: Bayesian Analysis of Single-Cell Sequencing Data. *PLOS Computational Biology*, **11**(6), e1004333.

Vallejos, C. A., Richardson, S., and Marioni, J. C. (2016). Beyond comparisons of means: Understanding changes in gene expression at the single-cell level. *Genome Biology*, **17**, 70.

van der Maaten, L. and Hinton, G. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, **9**(Nov), 2579–2605.

van Dijk, D., Sharma, R., Nainys, J., Yim, K., Kathail, P., Carr, A. J., Burdziak, C., Moon, K. R., Chaffer, C. L., Pattabiraman, D., Bierie, B., Mazutis, L., Wolf, G., Krishnaswamy, S., and Pe'er, D. (2018). Recovering Gene Interactions from Single-Cell Data Using Data Diffusion. *Cell*, **0**(0).

Villani, A.-C., Satija, R., Reynolds, G., Sarkizova, S., Shekhar, K., Fletcher, J., Griesbeck, M., Butler, A., Zheng, S., Lazo, S., Jardine, L., Dixon, D., Stephenson, E., Nilsson, E., Grundberg, I., McDonald, D., Filby, A., Li, W., Jager, P. L. D., Rozenblatt-Rosen, O., Lane, A. A., Haniffa, M., Regev, A., and Hacohen, N. (2017). Single-cell RNA-seq reveals new types of human blood dendritic cells, monocytes, and progenitors. *Science*, **356**(6335), eaah4573.

Virtanen, P., Gommers, R., Burovski, E., Oliphant, T. E., Cournapeau, D., Weckesser, W., alexbrc, Peterson, P., endolith, Stefan van der Walt, Mayorov, N., Wilson, J., Laxalde, D., Brett, M., Millman, J., Lars, eric-jones, Nelson, A., Kern, R., Moore, E., Leslie, T., Perktold, J., Carey, C., Feng, Y., Vanderplas, J., Haberland, M., cowlicks, Larson, E., Polat, I., and Reddy, T. (2018). Scipy/scipy: Scipy 1.1.0rc1.

Waskom, M., Botvinnik, O., O'Kane, D., Hobson, P., Lukauskas, S., Gemperline, D. C., Augspurger, T., Halchenko, Y., Cole, J. B., Warmenhoven, J., Julian de Ruiter, Pye, C., Hoyer, S., Vanderplas, J., Villalba, S., Kunter, G., Quintero, E., Bachant, P., Martin, M., Meyer, K., Miles, A., Ram, Y., Yarkoni, T., Williams, M. L., Evans, C., Fitzgerald, C., Brian, Fonnesbeck, C., Lee, A., and Qalieh, A. (2017). Mwaskom/seaborn: V0.8.1 (September 2017).

Wilf, H. S. (1999). *Generating Functionology*. Academic Press, Boston [u.a. OCLC: 247811549.

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., Ghodsi, A., Gonzalez, J., Shenker, S., and Stoica, I. (2016). Apache Spark: A Unified Engine for Big Data Processing. *Commun. ACM*, **59**(11), 56–65.

Zappia, L., Phipson, B., and Oshlack, A. (2017). Splatter: Simulation of single-cell RNA sequencing data. *Genome Biology*, **18**, 174.

Zeisel, A., Muñoz-Manchado, A. B., Codeluppi, S., Lönnerberg, P., La Manno, G., Juréus, A., Marques, S., Munguba, H., He, L., Betsholtz, C., Rolny, C., Castelo-Branco, G., Hjerling-Leffler, J., and Linnarsson, S. (2015). Brain structure. Cell types in the mouse cortex and hippocampus revealed by single-cell RNA-seq. *Science*, **347**(6226), 1138–1142.

Zhang, T., Ramakrishnan, R., and Livny, M. (1996). BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, SIGMOD '96, pages 103–114, New York, NY, USA. ACM.

Zheng, G. X. Y., Terry, J. M., Belgrader, P., Ryvkin, P., Bent, Z. W., Wilson, R., Ziraldo, S. B., Wheeler, T. D., McDermott, G. P., Zhu, J., Gregory, M. T., Shuga, J., Montesclaros, L., Underwood, J. G., Masquelier, D. A., Nishimura, S. Y., Schnall-Levin, M., Wyatt, P. W., Hindson, C. M., Bharadwaj, R., Wong, A., Ness, K. D., Beppu, L. W., Deeg, H. J., McFarland, C., Loeb, K. R., Valente, W. J., Ericson, N. G., Stevens, E. A., Radich, J. P., Mikkelsen, T. S., Hindson, B. J., and Bielas, J. H. (2017). Massively parallel digital transcriptional profiling of single cells. *Nat. Commun.*, **8**, 14049.