Title
Benchmarking Statistical and Machine-Learning Methods for Single-cell RNA Sequencing Data

Author
Xi, Nan

Publication Date
2021

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Benchmarking Statistical and Machine-Learning Methods for Single-cell RNA Sequencing Data

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Statistics

by

Nan Xi

2021

ABSTRACT OF THE DISSERTATION

Benchmarking Statistical and Machine-Learning Methods for Single-cell RNA Sequencing Data

by

Nan Xi

Doctor of Philosophy in Statistics

University of California, Los Angeles, 2021

Professor Jingyi Li, Chair

The large-scale, high-dimensional, and sparse single-cell RNA sequencing (scRNA-seq) data have raised great challenges in the pipeline of data analysis. A large number of statistical and machine learning methods have been developed to analyze scRNA-seq data and answer related scientific questions. Although different methods claim advantages in certain circumstances, it is difficult for users to select appropriate methods for their analysis tasks. Benchmark studies aim to provide recommendations for method selection based on an objective, accurate, and comprehensive comparison among cutting-edge methods. They can also offer suggestions for further methodological development through massive evaluations conducted on real data.

In Chapter 2, we conduct the first, systematic benchmark study of nine cutting-edge computational doublet-detection methods. In scRNA-seq, doublets form when two cells

are encapsulated into one reaction volume by chance. The existence of doublets, which appear as but are not real cells, is a key confounder in scRNA-seq data analysis. Computational methods have been developed to detect doublets in scRNA-seq data; however, the scRNA-seq field lacks a comprehensive benchmarking of these methods, making it difficult for researchers to choose an appropriate method for their specific analysis needs. Our benchmark study compares doublet-detection methods in terms of their detection accuracy under various experimental settings, impacts on downstream analyses, and computational efficiency. Our results show that existing methods exhibited diverse performance and distinct advantages in different aspects.

In Chapter 3, we develop an R package DoubletCollection to integrate the installation and execution of different doublet-detection methods. Traditional benchmark studies can be quickly out-of-date due to their static design and the rapid growth of available methods. DoubletCollection addresses this issue in benchmarking doublet-detection methods for scRNA-seq data. DoubletCollection provides a unified interface to perform and visualize downstream analysis after doublet-detection. Additionally, we created a protocol using DoubletCollection to execute and benchmark doublet-detection methods. This protocol can automatically accommodate new doublet-detection methods in the fast-growing scRNA-seq field.

In Chapter 4, we conduct the first comprehensive empirical study to explore the best modeling strategy for autoencoder-based imputation methods specific to scRNA-seq data. The autoencoder-based imputation method is a family of promising methods to denoise sparse scRNA-seq data; however, the design of autoencoders has not been formally discussed in the literature. Current autoencoder-based imputation methods either borrow

the practice from other fields or design the model on an ad hoc basis. We find that the method performance is sensitive to the key hyperparameter of autoencoders, including architecture, activation function, and regularization. Their optimal settings on scRNA-seq are largely different from those on other data types. Our results emphasize the importance of exploring hyperparameter space in such complex and flexible methods. Our work also points out the future direction of improving current methods.

The dissertation of Nan Xi is approved.

Ker-Chau Li

Yingnian Wu

Hongquan Xu

Jingyi Li, Committee Chair

University of California, Los Angeles

2021

TABLE OF CONTENTS

1 Introduction 1
2 Benchmarking Computational Doublet-Detection Methods for Single-cell RNA Sequencing Data 7
2.1 Introduction 7
2.2 Results 11
2.2.1 Doublet detection accuracy on real scRNA-seq datasets 11
2.2.2 Doublet detection accuracy on synthetic scRNA-seq data under various experimental settings and biological conditions 15
2.2.3 Effects of doublet detection on DE gene analysis 17
2.2.4 Effects of doublet detection on highly variable gene identification 19
2.2.5 Effects of doublet detection on cell clustering 20
2.2.6 Effects of doublet detection on cell trajectory inference 23
2.2.7 Performance of doublet-detection methods under distributed computing 25
2.2.8 Computational efficiency, scalability, stability, and software implementation of doublet-detection methods 27
2.3 Discussion 30
2.4 Methods 36
2.4.1 Real data preprocessing 36
2.4.2 Benchmark environment and parameter settings 38
2.4.3 Measures of doublet-detection accuracy 40
2.4.4 Simulation of scRNA-seq datasets containing doublets 41
2.4.5 Experimental settings used in benchmarking simulations 42
2.4.6 DE gene analysis 43
2.4.7 Identification of highly variable genes 44
2.4.8 Cell clustering analysis 45
2.4.9 Cell trajectory inference 46
2.4.10 Distributed computing 48
2.4.11 Scalability, stability, and usability 49
2.5 Acknowledgements 50

2.6 Figures and Tables 51
2.7 Supplementary Materials 64
2.7.1 Accuracy of computational doublet detection in relation to experimental techniques for doublet labeling 64
2.7.2 Pairwise similarities of computational doublet-detection methods 65
2.7.3 Comparison of hyperparameter selection in knn-based methods 66
3 An R Package for Benchmarking Computational Doublet-Detection Methods in Single-Cell RNA Sequencing Data Analysis 84
3.1 Introduction 84
3.2 A step-by-step protocol 85
3.2.1 Data download 85
3.2.2 Installation of DoubletCollection 87
3.2.3 Doublet detection accuracy on real scRNA-seq datasets 88
3.2.4 Hyperparameter tuning for doublet detection methods (Optional) 91
3.2.5 Doublet detection accuracy under various experimental settings and biological conditions 94
3.2.6 Effects of doublet detection on DE gene analysis 95
3.2.7 Effects of doublet detection on cell clustering 98
3.2.8 Effects of doublet detection on cell trajectory inference 100
3.2.9 Performance of doublet-detection methods under distributed computing 104
3.2.10 Computational aspects of doublet-detection methods (optional) 105
3.3 Expected Outcomes 106
3.4 Limitations 107
3.5 Troubleshooting 108
3.6 Figures 111
4 Benchmarking the Design of Deep Autoencoders for Denoising Single-Cell RNA Sequencing Data 116
4.1 Introduction 116
4.2 Results 120
4.2.1 Autoencoder for imputing scRNA-seq data 120
4.2.2 Three masking schemes for introducing missing values 122
4.2.3 Impact of autoencoder architecture on the imputation accuracy 123

4.2.4 Impact of activation function on the imputation accuracy 126
4.2.5 Impact of regularization on the imputation accuracy 128
4.2.6 Impact of autoencoder design on cell clustering 131
4.2.7 Impact of autoencoder design on DE gene analysis 134
4.3 Discussion 137
4.4 Methods 140
4.4.1 Data preprocessing and normalization 140
4.4.2 Training of autoencoders and imputation 141
4.4.3 Calculation of imputation normalized root MSE (NRMSE) 142
4.4.4 Activation functions 142
4.4.5 Cell clustering analysis 144
4.4.6 Simulation of synthetic scRNA-seq data 145
4.4.7 DE gene analysis 146
4.5 Figures 147
4.6 Supplementary Figures and Tables 154
5 Conclusions 169
References 172

LIST OF FIGURES

Chapter 2

1 Evaluation using real data 51

2 Evaluation using synthetic data, the effects on DE analysis, highly variable genes identification, and cell clustering 53

3 Effects on cell trajectory inference 55

4 Comparison on distributed computing, running time, scalability, and stability 57

5 A summary of benchmark results 59

S1 Comparison between DoubletDecon and other methods 68

S2 Cell clustering result by the DBSCAN, stability (AUROC), performance on different plantforms, and hyperparameter tuning 69

Chapter 3

1 Data repository with real and synthetic datasets 111

2 Evaluation on real data 112

3 Evaluation on synthetic data and the effects on DE analysis 113

4 Effects on cell clustering 114

5 Effects on cell trajectory inference and distributed computing 115

Chapter 4

1 Autoencoder and the measurement of imputation accuracy 147

2 The impact of depth and width 148

3 The impact of activation functions 149

4 The impact of weight decay regularization 150

5 The impact of dropout regularization 151

6 The impact of design on the cell clustering 152

7 The impact of design on the DE gene analysis 153

S1 The impact of depth and width (double exponential) 158

S2 The impact of depth and width (median) 159

S3 The impact of activation functions (double exponential) 160

S4 The impact of weight decay regularization (double exponential) 161
S5 The impact of dropout regularization (double exponential) 162
S6 The impact of design on the cell clustering (continued) 163
S7 The impact of design on the cell clustering (AMI) 164
S8 The impact of design on the DE gene analysis (precision) 165
S9 The impact of design on the DE gene analysis (recall, continued) 166
S10 The impact of design on the DE gene analysis (TNR) 167
S11 The seven activation functions evaluated 168

LIST OF TABLES

Chapter 2

1 An overview of nine computational doublet-detection methods 60

2 Real scRNA-seq datasets with experimentally annotated doublets 62

3 Usability of the nine doublet-detection methods 63

S1 AUPRC values of ten doublet-detection methods 71

S2 AUROC values of ten doublet-detection methods 72

S3 The number of outperforming baselines and the number of top-performing for each method on 16 benchmark scRNA-seq datasets 73

S4 Mean precision, recall, and TNR of ten doublet-detection methods 74

S5 Precision of doublets detection on 12 benchmark scRNA-seq datasets 75

S6 Recall of doublets detection on 12 benchmark scRNA-seq datasets 76

S7 TNR of doublets detection on 12 benchmark scRNA-seq datasets 77

S8 The number of identified doublets by DoubletDecon compared with the true number of doublets on 12 benchmark datasets 78

S9 Mean running time of nine doublet-detection methods and their AUPRCs on 16 benchmark scRNA-seq datasets 79

S10 Mean AUPRC values of eight doublet-detection methods on benchmark scRNA-seq datasets, categorized by four experimental techniques 80

S11 Mean Pearson correlation coefficient between every pair of doublet-detection methods in terms of their doublet scores across the 16 benchmark datasets 81

S12 Mean Jaccard index between every pair of doublet-detection methods in terms of their identified doublets 82

S13 The default hyperparameter settings of Scrublet and DoubletFinder 83

Chapter 4

S1 The datasets used to evaluate the imputation accuracy 154

S2 The datasets used to evaluate the cell clustering 155

S3 The datasets that generate synthetic data in DE gene analysis 157

ACKNOWLEDGMENTS

I owe a great deal of gratitude to my advisor, Professor Jingyi Jessica Li. She provides invaluable strategic advice at every stage of my graduate study at UCLA, from choosing a research topic to succeeding in the job market, as well as work-family balance. Working with her is among the most rewarding and pleasant experiences in my life. Nobody has a greater positive impact on my academic career than Professor Li. From her, I have learned how good an advisor can be and the enormous amounts of time and effort required to lead a student to grow up as a researcher. I hope to become the kind of professor to my students that Professor Li has been to her.

I would like to express my deep gratitude to my committee member, Professor Hongquan Xu, who has always been supportive of my study, research, and academic development. Professor Xu led me to Statistics. His academic rigor and enthusiasm always inspire me. I am grateful for his constant encouragement, patient guidance, and tremendous support throughout my graduate study at UCLA.

I am very fortunate to have Professor Yingnian Wu on my committee. His classes provide the backbone of the statistical training that I rely on in my career. Discussions with him during my graduate study are always encouraging and motivating. The mentorship from him gives me essential support in the academic job market.

I want to thank my committee member Professor Ker-Chau Li for the enormous insights he gives to help me become a researcher. His outstanding research and teaching extend my research area. I am deeply indebted to Professor Li for his invaluable advice on my dissertation. His great academic passion would always inspire me in my career.

I am very proud to work with my fellow students and junior researchers, especially Ruochen Jiang, Yiling Chen, Dongyuan Song, Tianyi Sun, Xinzhou Ge, Wei Li, Heather Zhou, Kexin Li, Yidan Sun, and all other members of the JSB group. Thanks for the help and advice they bring to my research and life. Their friendship and academic support make my time at UCLA much more pleasant. Finally, I am very grateful to my wife, Lin Wang, for her constant support. All of this would not have been possible without her undivided love and encouragement.

VITA

2003 — 2007	B.S. in Computer Science, Dalian University of Technology
2007 — 2009	Software Engineer, Panasonic
2010 — 2012	M.S. in Economics, Nankai University
2012 — 2015	Data Analyst, China Development Bank
2016 — 2021	Ph.D. Student and Candidate, Department of Statistics, University of California, Los Angeles

PUBLICATIONS

Xi, M.N. and Li, J.J. (2021). Benchmarking Computational Doublet-Detection Methods for Single-Cell RNA Sequencing Data. Cell Systems 12: 1-19.

Xi, M.N. and Li, J.J. (2021). Protocol for Benchmarking Computational doublet-Detection Methods in Single-Cell RNA Sequencing Data Analysis. STAR Protocols (Preprint).

Xi, N., Ma, D., Liou, M., Steinert-Threlkeld, Z., Anastasopoulos, J., and Joo, J. (2020). Understanding the Political Ideology of Legislators from Social Media Images. The International AAAI Conference on Web and Social Media (ICWSM).

Xi, N. and Joo, J. (2019). Face Attribute Dataset for Balanced Race (extended abstract). The Conference on Computer Vision and Pattern Recognition (CVPR) Workshop.

Xi, N. (2011). The Duopoly Analysis of Graphics Card Market. China Urban Economy 5: 64-65.

HONORS AND AWARDS

2021	Best Poster Award, Duke Industry Statistics Symposium
2020	FDA ORISE Fellowship
2012	CO-WIN Scholarship
2010, 2011, 2012	Graduate Scholarship Award, Nankai University
2006	Outstanding Performance Award in National Undergraduate Art Competition (Head of Tenors in University Chorus)
2004, 2005	Undergraduate Scholarship Award, Dalian University of Technology

CHAPTER 1

Introduction

Single-cell RNA sequencing (scRNA-seq) is a family of emerging sequencing technologies that have revolutionized biomedical sciences by revealing genome-wide gene expression levels within each of thousands to millions of individual cells [1–3]. Since its invention, scRNA-seq has become an essential experimental approach to investigate cell-to-cell heterogeneity, distinguish cell types and subtypes, identify cell-type-specific genes, and reveal cellular dynamic processes [4,5]. There are two major experimental techniques of scRNA-seq among many protocols and commercial platforms. The first approach is droplet microfluidics and well-based protocols [6,7]. This approach distributes a cell suspension into reaction volumes (droplets or wells) to hopefully encapsulate one cell per volume, and then mRNA molecules in each volume are labeled by a unique droplet barcode. Droplet microfluidics and well-based protocols have gained popularity because of their high throughput, low cost per cell, and ability to detect unique mRNA transcripts via unique molecular identifiers (UMIs) [8,9]. The second approach is plated-based protocol [10]. This approach sequences relatively small amounts of cells but with much deeper sequencing depths than droplet- and well-based protocols [11,12].

The recent development of experimental techniques in both sequencing approaches has significantly reduced the cost of scRNA-seq. As a result, many large-scale scRNA-seq datasets have been generated to answer a wide range of biological questions [13,14]. Each of those datasets may contain up to millions of cells and tens of thousands of genes per cell. The effective analysis of such large datasets requires a standardized pipeline, including quality control, normalization, batch effect correction, data integration, imputation and denoising, future selection, dimension reduction, cell clustering, differentially expressed (DE) gene analysis, cell trajectory inference, and visualization [15]. The diverse datasets, biological samples, and scientific questions in those analysis tasks post a great challenge but also an opportunity for computationalists to develop novel statistical and machine learning methods. At the time of writing, more than 900 computational methods are available for scRNA-seq data analysis, and many more are under development [16]. Researchers in the scRNA-seq community frequently face a choice among several, if not dozens of, different methods in one single data analysis task.

Benchmark studies aim to provide practitioners a clear guideline to choose appropriate computational methods for their specific scRNA-seq data analysis. A well-designed benchmark study systematically compares the performance of different methods based on real or synthetic scRNA-seq data with ground-truth biological labels. It should also explore the impacts of different methods on the downstream scRNA-seq data analysis. A benchmark study must be informative, comprehensive, accurate, unbiased, up-to-date, and reproducible. Based on those principles, we conducted the first, systematic benchmark study of computational doublet-detection methods for scRNA-seq data. We compared doublet-detection methods in terms of their detection accuracy under

various experimental settings, impacts on downstream analyses, and computational efficiency. Our results show that existing methods exhibit diverse performance and distinct advantages in different aspects. Our study provides much-needed guidance to researchers in choosing appropriate doublet-detection methods for scRNA-seq data analysis. Our results also point out directions for further methodological development and improvement in computational doublet detection, an active area of bioinformatics research [17].

The scRNA-seq community has conducted a number of well-designed benchmark studies on the aforementioned pipeline of scRNA-seq data analysis [18–23]. These studies are valuable resources that assist researchers in selecting appropriate computational methods. However, most of those benchmark studies are static, which makes them easily out-of-data due to the rapid growth of computational methods in the scRNA-seq field [24]. The recommendations from those studies need constant updating to catch up with the state-of-the-art status in method development. Because of the static design, updating benchmark studies is time-consuming and tedious, especially for experimentalists who lack programming skills. To address this issue for the doublet-detection benchmark study, we developed a statistical software DoubletCollection to automate the benchmark of doublet detection by integrating the installation and execution of cutting-edge doublet-detection methods [25]. DoubletCollection also provides a unified interface to perform and visualize downstream analysis after doublet-detection. DoubletCollection can automatically accommodate new doublet-detection methods and datasets in the fast-growing scRNA-seq field. Additionally, we created a protocol on how to use DoubletCollection in real-world applications. Our protocol defines a new paradigm to

execute and benchmark different doublet-detection methods, fine-tune their hyperparameters, and replicate the result in a transparent way.

The essential component of scRNA-seq data analysis is the gene expression matrix, in which each row represents one cell and each column represents one gene (or vice versa). ScRNA-seq data matrices are typically very sparse — the proportion of zero entries ranges from ~50% to ~95% [5]. Several factors contribute to such a high zero rate, including shallow sequencing depth (especially in droplet- and well-based protocols), the random noise introduced in the experimental process, and the biological absence of certain gene expressions [26]. The high sparsity of scRNA-seq data poses a great challenge to various downstream analyses due to the low signal-noise ratio [22]. The scRNA-seq community tries to address this issue by developing imputation methods to fill up zeros in data matrices. Currently, there are three broad categories of scRNA-seq imputation methods. The first is the model-based method that uses probabilistic models to identify and impute zeros introduced by technical variability [27,28]. The second is the data smoothing method that adjusts all expression values in one cell by using similar cells' gene expression [29,30]. The third is the data-reconstruction method that uses deep learning to obtain a latent space representation of the cells and then reconstructs a dense data matrix from the latent space representation [31,32]. Among more than 70 currently available imputation methods, autoencoder-based data-reconstruction methods have raised large attention due to their superior performance in several benchmark studies [22,33].

Despite their success in some applications, there is no formal discussion on the design of autoencoders in those imputation methods [22]. Current methods either borrow the experience learned from computer vision study or set up the autoencoder on an ad hoc

basis, which potentially limits the imputation performance on scRNA-seq data [24,34]. To address this issue, we conducted the first empirical study to explore the best modeling strategies in the design of autoencoders for imputing sparse scRNA-seq data. Specifically, we adjusted the three fundamental design aspects of autoencoders — architecture, activation function, and regularization, and examined their impacts on the overall imputation accuracy, downstream cell clustering, and DE gene analysis. Based on numerical experiments on large-scale real and synthetic scRNA-seq datasets, we find that different from current practice using shallow and wide neural networks with ReLU activation functions [35], deep and narrow neural works with sigmoid or tanh activation functions [36] provide better imputation accuracy, cell clustering, and DE gene analysis. In terms of regularization, weight decay [37] significantly improves the cell clustering and DE gene analysis, while dropout [38] has moderate improvement on the overall imputation accuracy. Our findings suggest a unique modeling strategy suitable for scRNA-seq data. Our results also point out directions for future methodological development and hyperparameter tuning for currently available methods.

Our work has three broad impacts on the field of scRNA-seq data analysis. First, our benchmark study is the first one that provides practitioners an objective, comprehensive, and state-of-the-art recommendation on the selection of computational doublet-detection methods. Second, our statistical software DoubletCollection is the first framework that standardizes the installation, execution, and benchmark of cutting-edge doublet-detection methods. Third, we find that the autoencoder requires a unique design for imputing scRNA-seq data, which is largely different from its applications in other fields. This dissertation is organized in the following order: In chapter 2, we introduce benchmarking

computational doublet-detection methods for scRNA-seq data; In chapter 3, we introduce statistical software DoubletCollection and the protocol for benchmarking computational doublet-detection methods in scRNA-seq data analysis; In chapter 4, we introduce designing deep autoencoders to denoise scRNA-seq Data; In chapter 5, we summarize our work and discuss some future research questions.

CHAPTER 2

Benchmarking Computational Doublet-Detection Methods for Single-cell RNA Sequencing Data

2.1 Introduction

In scRNA-seq, two major experimental protocols — droplet microfluidics and well-based protocols — distribute a cell suspension into reaction volumes (droplets or wells) to hopefully encapsulate one cell per volume (i.e., a singlet), and then mRNA molecules in each volume are labeled by a unique droplet barcode. For simplicity, we will refer to a reaction volume as a droplet in the following text. During the distribution step, however, one droplet may encapsulate more than one cell, creating a so-called doublet that is disguised as a single cell [5]. The doublet rate (i.e., the proportion of doublets) in a scRNA-seq experiment depends on the throughput and protocol, and doublets may constitute as many as 40% of droplets [39]. There are two major classes of doublets: homotypic doublets, which are formed by transcriptionally similar cells, and heterotypic doublets, which are formed by cells of distinct types, lineages, or states [40,41]. Compared with homotypic

doublets, heterotypic doublets are generally easier to detect due to their distinct gene expression profiles unlike those of singlets [41].

The existence of doublets, especially heterotypic doublets, in scRNA-seq datasets may confound downstream analysis; for example, doublets can form spurious cell clusters, interfere with differentially expressed (DE) gene analysis, and obscure the inference of cell developmental trajectories [5,40]. Several experimental techniques have been developed to detect doublets in scRNA-seq using droplet barcodes. Example techniques include cell hashing (doublets are the droplets whose barcodes are associated with more than one oligo-tagged antibody) [42], species mixture (doublets are the droplets whose barcodes are associated with more than one species) [40], demuxlet (doublets are the droplets whose barcodes are associated with mutually exclusive sets of SNPs) [43], and MULTI-seq (doublets are the droplets whose barcodes are associated with more than one lipid-tagged index) [44]. However, these techniques require special experimental preparation, extra costs, and time, and they are not guaranteed to remove all doublets, e.g., demuxlet cannot detect the doublets formed by cells from the same individual. Moreover, they cannot remove doublets from existing scRNA-seq data.

Realizing the limitations of experimental strategies, researchers have attempted to tackle this doublet challenge from an alternative perspective: developing computational methods to detect doublets from already-generated scRNA-seq data [5]. So far, nine doublet-detection methods have been developed (with software packages and full-text manuscripts) based on distinct algorithmic designs [39–41,45–48] (Table 1). Here is a brief summary of these methods except hybrid, which is a combination of two methods: bcds and cxds. Seven out of the eight methods (with cxds as the only exception) first generate

artificial doublets by combining gene expression profiles of two randomly selected droplets. Except for DoubletDecon, the other six methods subsequently define a doublet score for each original droplet as the level of similarity the droplet has to those artificial doublets; next, with a pre-defined or user-specified threshold, they detect doublets as the original droplets whose doublet scores exceed the threshold. The key difference of the seven artificial-doublet-based methods is how they distinguish original droplets from artificial doublets: five of them use classification algorithms (Scrublet, doubletCells, and DoubletFinder use *k*-nearest neighbors (kNN); bcds uses gradient boosting; Solo uses neural networks), DoubletDetection uses the hypergeometric test, and DoubletDecon decides whether an original droplet resembles an artificial doublet based on its deconvolution algorithm (unlike the other methods, DoubletDecon identifies doublets without providing doublet scores). As the only method that does not generate artificial doublets, cxds defines doublet scores based on gene co-expression, and similar to the other six doublet-score-based methods, it subsequently thresholds doublet scores to identify doublets. While each method was shown to perform well under certain metrics by its developers, currently, there is no systematic, third-party benchmarking of these methods' doublet-detection accuracy, effects on downstream analysis, or computation efficiency. As a result, users lack guidelines to choose an appropriate doublet-detection method for their analysis task. Hence, a detailed assessment of existing doublet-detection methods is in great demand. In addition to assisting users, it will provide useful guidance for computationalists to improve existing methods or develop new methods.

Here, we conducted the first comprehensive benchmark study of computational methods for doublet detection. We evaluated nine cutting-edge methods—doubletCells

[48], Scrublet [40], cxds [45], bcds [45], hybrid [45], Solo [39], DoubletDetection [47], DoubletFinder [41], and DoubletDecon [46]—in three aspects. First, we compared their overall doublet detection accuracy using two criteria: the area under the precision-recall curve (AUPRC) and the area under the receiver operating characteristic curve (AUROC), on a collection of 16 real scRNA-seq datasets containing experimentally annotated doublets. To further evaluate the performance of these methods under various experimental settings, we simulated 80 realistic scRNA-seq datasets and evaluated the AUPRC and AUROC of each method under a wide range of doublet rates, sequencing depths, numbers of cell types, and cell-type heterogeneity levels. Second, considering that the ultimate goal of doublet detection is to improve the accuracy of downstream scRNA-seq data analyses, we compared these nine doublet-detection methods in terms of their impacts on three downstream analyses: cell clustering, DE gene analysis, and cell trajectory inference. We simulated seven doublet-containing scRNA-seq datasets with pre-specified cell types, DE genes, and cell trajectories. Then we evaluated the accuracy of the three downstream analyses by their state-of-the-art computational methods before and after doublets were removed by each doublet-detection method. The rationale is that a good doublet-detection method should improve the accuracy of downstream analyses after its use. Third, we compared the computational efficiency of doublet-detection methods in aspects including distributed computing, speed, scalability, stability, and usability.

In summary, the nine doublet-detection methods exhibited a large variation in their performance under each evaluation criterion. First, the benchmarking result of detection accuracy shows that there is still room for improvement: the best method DoubletFinder achieved a mean AUPRC value of 0.537 on 16 real datasets (Table S1). On simulated

datasets, most methods performed better on datasets with higher doublet rates, larger sequencing depths, more cell types, or greater heterogeneity between cell types. Second, we observed that doublet removal by most methods indeed improved the elimination of spurious cell clusters, the identification of DE genes, and the inference of cell trajectories, yet the degree of improvement varied from method to method. Third, most methods except cxds had deteriorated performance under distributed computing because global data information was lost in each distributed data batch. The cxds method also performed the best in terms of speed and scalability. Overall, DoubletFinder is highlighted as the best computational doublet-detection method for its highest detection accuracy and largest improvement on downstream analyses, while cxds is found as the most computationally efficient method in our benchmark.

2.2 Results

2.2.1 Doublet detection accuracy on real scRNA-seq datasets

To evaluate the overall doublet detection accuracy of the nine methods, we collected 16 public scRNA-seq datasets with doublets annotated by experimental techniques [40,42–44] (Methods). Our collection covers a variety of cell types, droplet and gene numbers, doublet rates, and sequencing depths, thus representing varying levels of difficulty in detecting doublets from scRNA-seq data (Table 2). To the best of our knowledge, our collection is by far the most comprehensive set of scRNA-seq data that contains experimentally validated doublets, and it can serve as a benchmark standard for future method development.

To benchmark the nine methods, we included two baseline methods, which simply use the library size (lsize) and the number of expressed genes (ngene) of each droplet as their respective doublet detection criterion [5,40]. Except for DoubletDecon, all the methods output a doublet score for each droplet (Table 1; the two baseline methods have lsize and ngene as their doublet scores; a droplet with a larger score is more likely a doublet), and we define their detection accuracy as their AUPRC and AUROC values (Methods). We found that all the methods successfully output their identified doublets from all the 16 datasets except DoubletDetection, which could not run on the pdx-MULTI dataset. Across the 16 datasets, each method exhibited a large variance in its detection accuracy, and no method consistently achieved the top performance (Figure 1a–b; Supplementary Tables S1-S2). Compared with the two baseline methods, doubletCells is the only method that did not outperform them on a majority of datasets, while Solo and hybrid are the only two methods that consistently outperformed them on all datasets (Supplementary Table S3). Overall, DoubletFinder and Solo achieved the highest mean AUPRC and AUROC values across datasets, respectively (Supplementary Tables S1–S2). DoubletFinder was also the top-performing method on the most datasets in terms of both AUPRC and AUROC (Supplementary Table S3). We note that all the methods had AUPRC values much lower than their AUROC values on every dataset, an expected phenomenon given the imbalance between the number of singlets and doublets. Since AUROC is an overly optimistic measure of accuracy under such imbalanced scenarios [49], we will focus on AUPRC in the following discussion.

The highest AUPRC value on each dataset ranges from 0.239 to 1.000, with a mean of 0.570 across the 16 datasets (Supplementary Table S1). This large discrepancy

between datasets is further exemplified by the fact that several methods achieved almost perfect AUPRC values on two datasets: hm-12k and hm-6k, while all the methods performed poorly on another two datasets: pbmc-1B-dm and J293t-dm (with AUPRC values under 0.335). A likely reason for this discrepancy is how doublets are annotated in these real datasets. In hm-12k and hm-6k, doublets are annotated as the droplets that contain cells of two species, so all doublets are heterotypic and easy to identify [39–41,45]. In contrast, doublets annotated in the other datasets may include homotypic doublets that are difficult to identify, posing a challenge to doublet-detection methods; or they may miss certain heterotypic doublets (e.g., if doublets are defined as the droplets that contain cells from two individuals, then heterotypic doublets formed by cells of different types within an individual would be missed), creating a downward bias in the calculation of detection accuracy (see further discussion in the Supplementary). In addition, varied data quality and cell heterogeneity pose different levels of difficulty to doublet detection. The highest mean AUPRC value, which was achieved by DoubletFinder, is only 0.537. These results demonstrate the general difficulty in detecting doublets from scRNA-seq data and suggest possible room for improvement by future method development.

Motivated by the fact that doublets are identified based on a single threshold in practice, we further examined the detection accuracy of doublet-detection methods under a specific identification rate, i.e., the percentage of droplets identified as doublets. For each method, the top 10%, 20%, and 40% droplets with the highest doublet scores were identified as doublets, and the corresponding precision, recall, and true negative rates (TNRs) were calculated (Figure 1c; Supplementary Table S4). As expected, higher identification rates led to higher recall and lower TNR values. Interestingly, the precision

decreased as the identification rate increased, a phenomenon suggesting that all doublet detection methods tend to assign higher doublet scores to annotated doublets and thus desirable (Figure 1c). The comparison of doublet-detection methods gave a result consistent with that based on the overall detection accuracy measures AUPRC and AUROC. DoubletFinder and Solo were still the top two methods in terms of the mean precision, recall, and TNR, where the mean was calculated across the 16 datasets (Supplementary Table S4).

Since DoubletDecon cannot output doublet scores, we could not calculate its AUPRC or AUROC on a dataset and thus excluded it from the previous comparison. To fairly compare DoubletDecon with other methods, we ran DoubletDecon on every dataset and recorded its number of identified doublets if successful; then we thresholded the doublet scores of other methods so that they identified the same number of doublets as DoubletDecon did. Based on the resulting doublets identified by each method from every dataset, we calculated the precision, recall, and TNR (Methods). By these three criteria, DoubletDecon and doubletCells did not outperform the baseline methods lsize and ngene. Among the other seven methods, Solo and DoubletFinder achieved the highest precision and TNRs, while Solo and hybrid obtained the highest recall rates (Supplementary Figure S1a and Tables S5–S7). Moreover, we observed that DoubletDecon failed to run on four datasets (hm-12k, pbmc-2ctrl-dm, J293t-dm, and nuc-MULTI) and tended to overestimate the number of doublets (Supplementary Table S8). Our results suggest that DoubletDecon needs improvement in its accuracy and robustness. Adding the functionality that outputs doublet scores will also enhance the usability of DoubletDecon,

because users can then have the flexibility to decide the number of doublets to be detected and removed based on their preference and knowledge [50].

2.2.2 Doublet detection accuracy on synthetic scRNA-seq data under various experimental settings and biological conditions

To thoroughly evaluate the performance of doublet-detection methods under a wide range of experimental settings and biological conditions, we utilized scDesign [51,157], a statistical simulator that generates realistic scRNA-seq datasets well mimicking real data generated by a variety of scRNA-seq experimental protocols. It is advantageous to use synthetic data to benchmark doublet-detection methods because we would have the access to ground-truth doublets and the flexibility to vary experimental settings and biological conditions in a comprehensive way. Specifically, we generated 80 scRNA-seq datasets with varying doublet rates (i.e., percentages of doublets), sequencing depths, cell types, and between-cell-type heterogeneity levels (Methods). Except for DoubletDecon, we applied every doublet-detection method to all these synthetic datasets and calculated its AUPRC values to measure its accuracy. Figure 2a shows how the performance of every method changed as we varied the doublet rate, the sequencing depth, the number of cell types, or the between-cell-type heterogeneity level. First, all the eight methods had improved accuracy as the doublet rate increased. This result is not surprising, as these methods all formulated the doublet detection problem, explicitly or implicitly, as a binary classification problem where the two classes are singlets and doublets. The more balanced the two classes are in size, the easier the binary classification is, in general. Given the fact that, under both droplet microfluidics and well-based scRNA-seq protocols,

doublets are more likely to form as the number of cells increases [5,40,52], our result suggests that doublet-detection methods would work more effectively on scRNA-seq datasets with more cells (or droplets). This finding agrees with our previous result that all the methods performed the worst on the J293t-dm dataset, which contains only 500 droplets, the fewest among all the 16 datasets. Second, we found that the performance of these methods consistently benefited from a larger sequencing depth. This is in line with the expectation that deeper sequencing creates a higher data resolution, making doublet-detection methods more capable of differentiating doublets from singlets. Third, we evaluated the impact of the number of cell types on the accuracy of doublet-detection methods. It is expected that a cell mixture with more cell types would result in more heterotypic doublets, which are formed by cells of different types. Thanks to their distinct gene expression profiles that do not resemble those of any cell types, heterotypic doublets are, in general, easier to detect than homotypic doublets, which are formed by cells of the same type [40]. As expected, most methods exhibited improved accuracy as the number of cell types increased, with cxds, bcds, and hybrid (a combination of cxds and bcds) as the only three exceptions. Fourth, we investigated how the between-cell-type heterogeneity level—the extent to which gene expression profiles differ between cell types—would affect the accuracy of doublet detection. In theory, the greater the heterogeneity, the more distinct are heterotypic doublets from singlets. Again, all the methods fit this theory except cxds, bcds, and hybrid. Hence, we saw consistent results about the effects of the number of cell types and the between-cell-type heterogeneity level on doublet detection.

We also compared the AUROC values of the eight doublet-detection methods on the same synthetic scRNA-seq datasets as above (Supplementary Figure S1b). Consistent with our AUPRC results, most methods performed better on the datasets with a higher doublet rate, a larger sequencing depth, more cell types, or a greater level of between-cell-type heterogeneity, though the improvement in AUROC was less significant than in AUPRC. This is expected as AUPRC is a better accuracy measure than AUROC for imbalanced binary classification [53]. Combining our AUPRC and AUROC results, we found DoubletFinder as the top-performing method across all the experimental settings and biological conditions we studied. DoubletDetection and Scrublet also demonstrated strong performance compared with the rest of methods. We excluded DoubletDecon from this comparison and the following cell clustering, DE gene identification, and cell trajectory inference analyses because it failed to run on most of our synthetic datasets, likely due to its software implementation issue [54].

2.2.3 Effects of doublet detection on DE gene analysis

The existence of doublets in scRNA-seq datasets is expected to confound the downstream DE gene analysis by violating the necessary “identical distribution” assumption (i.e., cells of the same type follow the same distribution of gene expression levels) in statistical tests [5]. As a result, if a doublet-detection method is effective, its doublet removal should improve the accuracy of DE gene analysis. To evaluate the eight doublet-detection methods from this perspective, we used scDesign to generate a synthetic scRNA-seq dataset with two cell types and 1126 between-cell-type DE genes (6% of a total of 18760 genes; Methods). We referred to this dataset as the “clean data.” We then mixed each cell type with randomly forming doublets by targeting a 40% doublet

rate, and the resulting dataset was referred to as the “contaminated data.” Next, we applied each doublet-detection method to the dataset and removed 40% droplets (with the highest doublet scores assigned by each method) from the contaminated data. Finally, we conducted DE gene analysis using three methods—DESeq2 [55], MAST [56], and Wilcoxon rank-sum test [57]—on the clean data, the contaminated data, and the dataset after each doublet-detection method was applied. The DE gene analysis result was summarized in three accuracy measures: precision, recall, and TNR, all of which were calculated under the Bonferroni-corrected p-value threshold of 0.05, the default threshold used by DESeq2 and MAST [58]. We benchmarked the accuracy resulted from each doublet-detection method against the negative control (the accuracy based on the contaminated data) and the positive control (the accuracy based on the clean data). Figure 2b shows that all the three DE methods achieved extremely high precision (> 98%) and TNRs (> 97%) even on the contaminated data, an expected result because these DE methods all utilize statistical tests and are inherently conservative in their identification of DE genes. Such conservativeness makes these DE methods only identify the genes that are highly likely DE, leading to high precision (the percentage of true DE genes among the identified genes) and TNR (the percentage of non-identified genes among the true non-DE genes). Although the TNR result seems counterintuitive as the TNR values after doublet detection and removal even exceeded the TNR values of the clean data by around 0.005, this difference was merely due to the statistical uncertainty of these TNR values and thus not conclusive. On the other hand, recall (the percentage of identified genes among the true DE genes) is an informative measure that reflects the negative influence of doublets: for all the three DE methods, their recall dropped from ~70% on the

clean data to ~63% on the contaminated data. Pleasantly, all the eight doublet-detection methods were effective in improving the recall (Figure 2c). In particular,DoubletFinder, doubletCells, bcds, and hybrid consistently had top performance regardless of the choice of DE methods. This result confirms that removing doublets is indeed beneficial for DE gene analysis.

2.2.4 Effects of doublet detection on highly variable gene identification

The identification of highly variable genes (HVGs) is an essential step that precedes cell dimension reduction, cell clustering, and cell trajectory inference in scRNA-seq data analysis [59]. The goal of this step is to identify HVGs, i.e., the informative genes that exhibit strong cell-to-cell variations and thus can distinguish cells, so that the dimensions of each cell can be reduced from tens of thousands of genes to thousands, or even hundreds of genes, to facilitate those downstream analyses. Considering the importance of HVG identification, we evaluated the extent to which the identification would be negatively affected by doublets [60] and how much the eight doublet-detection methods could alleviate such negative impacts. For this purpose, we simulated a clean scRNA-seq dataset without doublets by scDesign, and then we added randomly formed doublets to generate three contaminated datasets with 10%, 20%, and 40% doublet rates. For each contaminated dataset, we applied the eight doublet-detection methods to remove a percentage of droplets that received the highest doublet scores, and the percentage was set as the dataset's doublet rate. As a result, each contaminated dataset corresponds to eight post-doublet-detection datasets. Then we used Seurat [61,62] to identify HVGs from the clean dataset, the three contaminated datasets, and the 24 post-doublet-detection datasets. We refer to the identification results as a set of clean HVGs, three sets of

contaminated HVGs, and 24 sets of post-doublet-detection HVGs. An effective doublet-detection method is expected to result in post-doublet-detection HVGs that agree better with the clean HVGs than the corresponding contaminated HVGs do. To measure the agreement between two sets of HVGs, we used the Jaccard index, which is the ratio of the size of the intersection to the size of the union of the two sets. The larger the Jaccard index, the better agreement the two sets have. In our evaluation, for each doublet rate, the Jaccard index between the contaminated HVGs and the clean HVGs served as the negative control. Figure 2d shows that the negative control Jaccard index decreased from 0.772 to 0.447 as the doublet rate increased from 10% to 40%, matching our expectation. Among the eight doublet-detection methods, DoubletFinder and Scrublet were the only two methods whose post-doublet-detection HVGs consistently led to better Jaccard indices than the negative controls under all three doublet rates. Notably, the benefit of doublet detection on HVG identification was most obvious at the 40% doublet rate, under which all the doublet-detection methods outperformed the negative control.

2.2.5 Effects of doublet detection on cell clustering

Another major motivation to remove doublets from scRNA-seq data is to avoid the misinterpretation of spurious cell clusters (i.e., droplet clusters) formed by heterotypic doublets as novel cell types [5,40]. To evaluate the capacity of doublet-detection methods for removing spurious cell clusters, we used scDesign to simulate realistic scRNA-seq datasets composed of four, six, or eight cell types and mixed with 20% randomly forming doublets (i.e., the true doublet rate is 20%). We performed cell clustering on each of these datasets after we applied every doublet-detection method and removed a certain percent of droplets that received the highest doublet scores from that method (Methods).

Considering that the true doublet rate is unknown and difficult to estimate in practice, we varied this removal percentage from 0% to 25%, with a step size of 1%. For the subsequent cell clustering, we followed the most popular Seurat method to apply the Louvain clustering algorithm [63], which automatically determines the number of cell clusters in a data-driven way. Then for each dataset, every doublet-detection method, and each removal percentage, we compared the number of cell clusters with the number of cell types. Figure 2e shows that, under the ideal scenario that the removal percentage was set to the true doublet rate 20%, four methods (Scrublet, Solo, DoubletDetection, and DoubletFinder) consistently removed spurious cell clusters and led to the correct numbers of cell types. Among the eight methods, DoubletDetection and DoubletFinder exhibited the most robust performance, as they successfully led to the correct numbers of cell types under the widest range of removal percentages. Scrublet and Solo also exhibited good performance in removing spurious cell clusters. In contrast, doubletCells, cxds, bcds, and hybrid all had unstable performance, and they did not always remove spurious cell clusters even under the ideal scenario (when the removal percentage was set to 20%). Overall, this result supports the use of DoubletDetection and DoubletFinder to remove doublets before the application of cell clustering to identify novel cell types.

Unlike heterotypic doublets, homotypic doublets do not form spurious clusters because of their similar gene expression profiles to those of singlets of the same cell type [40]. In other words, homotypic doublets tend to cluster together with singlets. Even though the existence of homotypic doublets does not much affect cell clustering, it may potentially bias the identification of cell-type-specific genes by DE gene analysis because homotypic doublets are not real cells. To evaluate the capacity of doublet-detection methods in

eliminating homotypic doublets, we calculated the proportion of singlets in each identified cell cluster when the number of cell clusters matched the number of cell types in Figure 2e (Methods). Figure 2f shows that Scrublet led to cell clusters with the highest proportions of singlets. DoubletDetection and DoubletFinder also had excellent performance, and these three methods all clearly outperformed the rest of the methods. Combining the results in Figure 2e–f, we conclude that Scrublet, DoubletDetection, and DoubletFinder demonstrated the best capacity in removing heterotypic and homotypic doublets.

To examine how robust the above results are to the choice of clustering algorithms, we repeated the above analyses using a second clustering algorithm: the density-based spatial clustering of applications with noise (DBSCAN) [64]. Compared with the Louvain clustering algorithm, the DBSCAN algorithm led to the correct numbers of cell clusters under fewer and more sporadic removal percentages for all the doublet-detection methods (Supplementary Figure S2a). This result suggests that the DBSCAN algorithm works less effectively than the Louvain algorithm for clustering cells in scRNA-seq data [19,65]. Nevertheless, with the DBSCAN algorithm, Scrublet, DoubletDetection, and DoubletFinder still achieved the top performance in removing spurious cell clusters and homotypic doublets (Supplementary Figure S2a–b). In summary, based on the results of two clustering algorithms, we would recommend DoubletDetection and DoubletFinder as the top two choices for removing spurious cell clusters in cell clustering analysis, and we identified Scrublet and DoubletFinder as the best-performing algorithms for removing homotypic doublets before the identification of cell-type-specific genes.

2.2.6 Effects of doublet detection on cell trajectory inference

Another important scRNA-seq data analysis is to infer a cell trajectory, which corresponds to a cellular process such as cell differentiation, immune responses, and carcinogenesis, based on the similarity of cells in terms of gene expression profiles [20]. An inferred cell trajectory is called pseudotime, an ordering of cells in a path or a tree [66,158]. The accuracy of cell trajectory inference depends on both the inference methods and the scRNA-seq data quality. Similar to cell clustering, cell trajectory inference is also biased by the existence of doublets [18]. In particular, heterotypic doublets may result in spurious branches in an inferred trajectory. We expect that doublet-detection methods, if effective, should increase the accuracy of cell trajectory inference. To evaluate the eight doublet-detection methods from this perspective, we used Splatter [67] to generate two scRNA-seq datasets: one including a bifurcating trajectory and the other containing a conjunction of three sequential trajectories (Methods). We referred to them as the “clean data.” Then we mixed the two datasets with randomly forming doublets by targeting a 20% doublet rate, and the resulting datasets were referred to as the “contaminated data.” Similar to our DE gene analysis, we used each doublet-detection method to remove 20% droplets (with the highest doublet scores assigned by that method) from each contaminated dataset. As a result, we obtained two suites of datasets corresponding to a bifurcating trajectory and a conjunction of three sequential trajectories, with each suite containing the clean data, the contaminated data, and the data cleaned by each doublet-detection method. For cell trajectory inference, we applied Slingshot [68] to the first suite of datasets (Figure 3a) and minimum spanning tree (MST) [69] to the second suite of datasets (Figure 3b). We chose Slingshot and MST because they were the top-performing methods in previous

benchmark studies [18,20]. We considered the cell trajectories inferred from the clean data and the contaminated data as the positive and negative controls, respectively. Figure 3a–b shows that the doublets in the contaminated data indeed led to spurious branches that did not exist in the inferred trajectories from the clean data. Except for doubletCells, all the doublet-detection methods effectively removed doublets such that spurious branches no longer existed in the inferred cell trajectories. In particular, in the second task of inferring a conjunction of three sequential trajectories (Figure 3b), Scrublet, DoubletDetection, and DoubletFinder led to inferred trajectories that most resembled the trajectory inferred from the clean data. Figure 3a–b also shows that DoubletDetection and DoubletFinder are the best two methods for removing the "outlier" doublets whose gene expression profiles do not resemble those of any singlets.

Following cell trajectory inference, a typical next step is to explore gene expression dynamics along the inferred trajectory and to identify temporally DE genes [5,20]. Hence, the accuracy of cell trajectory inference largely determines the accuracy of temporally DE gene identification. Beyond checking the inferred cell trajectories after doublet removal as in Figure 3a–b, we evaluated the effects of doublet removal on the identification of temporally DE genes. We used Splatter to simulate a scRNA-seq dataset with a single lineage and 250 temporally DE genes out of a total of 750 genes (Methods). We referred to this dataset as the "clean data." We then mixed the data with randomly forming doublets by targeting a 20% doublet rate, and the resulting dataset was referred to as the "contaminated data." Next, we used eight doublet-detection methods to remove 20% droplets (with the highest doublet scores assigned by each method) from the contaminated data. Finally, we employed a general additive model (GAM) [70] to regress

each gene's expression levels on the corresponding cell/droplet pseudotime inferred by Slingshot or TSCAN [71] on the clean data, the contaminated data, and the dataset after each doublet-detection method was applied. Note that we replaced MST by TSCAN because MST does not output pseudotime values for droplets and TSCAN is built upon the MST algorithm. The temporally DE gene analysis result was summarized in three accuracy measures: precision, recall, and TNR, all of which were calculated under the Bonferroni-corrected p-value threshold of 0.05. Again, we used the accuracy obtained from the clean data and the contaminated data as the positive and negative controls, respectively. Doublet removal made a more significant improvement on the identification of temporally DE genes when Slingshot was used for trajectory inference (Figure 3c–d). With Slingshot, all the eight doublet-detection methods except doubletCells successfully restored the precision, recall, and TNR from low values on the contaminated data to values as high as those on the clean data. With TSCAN, however, the restoration effects were only obvious in precision and TNR by Solo and cxds. In summary, doublet removal is beneficial for cell trajectory inference and the subsequent identification of temporally DE genes, and we observed strong beneficial effects when Slingshot was used for trajectory inference.

2.2.7 Performance of doublet-detection methods under distributed computing

A grand challenge in single-cell data sciences is the skyrocketing demand for computational and storage resources due to the rapidly increasing data sizes [24]. For example, a scRNA-seq dataset may contain up to millions of droplets, each of which has

expression levels of tens of thousands of genes [13]. Analyzing such huge datasets is often beyond the capacity of a single computer but requires distributed computing, which analyzes data subsets in parallel. Specific to the doublet-detection task, distributed computing means that droplets are divided into batches, one batch per computer node, due to massive data sizes or limited computational capacity; then a doublet-detection method would be applied separately to assigning doublet scores to droplets in each batch. After this parallelization step, doublet scores would be pooled from multiple batches, and a threshold would be set on the pooled doublet scores to detect doublets. Compared with the centralized computing that uses all the droplets together, distributed computing may have deteriorated doublet-detection accuracy due to the limited data information within each droplet batch. Hence, how a doublet-detection method performs under distributed computing is an important evaluation criterion for the scalability and flexibility of the method.

To investigate the performance of doublet-detection methods under distributed computing, we randomly divided two large real scRNA-seq datasets—pbmc-ch and pbmc-2ctrl-dm—into a varying number of batches with equal numbers of droplets, and we evaluated how the doublet-detection accuracy of each method changed with the number of batches. It is expected that the more batches, the worse the accuracy, and our results confirmed this. Figure 4a–b shows the AUPRC and AUROC values of each method under each number of batches, which varied from 1 to 10. The AUPRC and AUROC values were calculated based on the pooled doublet scores as described above. We excluded DoubletDecon from this comparison because it failed to run for most numbers of batches, again suggesting its software implementation issue [54]. With only one

batch, distributed computing is reduced to centralized computing, and the corresponding accuracy is supposedly the performance ceiling of every method. As expected, most doublet-detection methods had decreasing accuracy, which is more clear in AUPRC (Figure 4a) than AUROC (Figure 4b), as the number of batches increased. Among the eight methods, doubletCells is an underperforming outlier with the lowest overall accuracy. DoubletDetection and Solo are among the top-performing methods under centralized computing; however, they exhibited the largest accuracy decrease under distributed computing. In contrast, DoubletFinder is consistently a top performer, demonstrating its superior accuracy again and its robustness under distributed computing.

2.2.8 Computational efficiency, scalability, stability, and software implementation of doublet-detection methods

In addition to the above evaluation that focused on the effects of doublet removal on various scRNA-seq data analyses, we also compared doublet-detection methods in four computational aspects: efficiency, scalability, stability, and software implementation. First, we summarized the running time of the nine doublet-detection methods (including their required data preprocessing steps; Methods) on the 16 real scRNA-seq datasets in Table 2. Figure 4c shows that cxds is the fastest method, while Solo, DoubletDecon, DoubletDetection, and DoubletFinder are significantly slower than the other methods. Figure 4d shows that there was no straightforward relationship between the mean AUPRC and the mean running time of eight doublet-detection methods (with the mean calculated across the 16 real datasets). Nevertheless, the three most computationally intensive methods—Solo, DoubletDetection, and DoubletFinder—had better accuracy

than the other methods except hybrid did. Interestingly, the hybrid method, an ensemble of cxds and bcds, largely improved on its both base methods without much running time increase. Among all methods, DoubletFinder achieved the highest mean AUPRC while not being the most computationally intensive method. Normalizing the mean running time by the mean AUPRC value for every method, we found cxds as the most resource-efficient method (Supplementary Table S9).

Second, we examined the scalability of doublet-detection methods by showing how fast their running time increases as the number of droplets grows. We used scDesign to generate 25 synthetic scRNA-seq datasets with the number of droplets ranging from 400 to 10,000 (Methods). Then we applied each doublet-detection method to these datasets and recorded its running time. (DoubletDecon was excluded because it failed to run on most synthetic data.) As shown in Figure 4e, all methods except Solo had running time scaled linearly with the number of droplets. The reason that Solo exhibited an erratic relationship between its running time and the number of droplets is probably due to its neural-network design. Among the other seven methods, cxds and DoubletDetection demonstrated the best and worst scalability, respectively.

Third, we evaluated doublet-detection methods in terms of the statistical stability, i.e., how much their AUPRC and AUROC values vary across subsets of droplets and genes. The smaller the variation, the larger the statistical stability. We randomly downsampled two large real scRNA-seq datasets—pbmc-ch and pbmc-2ctrl-dm—into 20 data subsets with 90% droplets and 90% genes. Then we applied each doublet-detection method to these data subsets and recorded the resulting AUPRC and AUROC values. (DoubletDecon was excluded because we were unable to calculate its AUPRC and

AUROC values, as explained before.) Figures 4f and S2c show the distributions of AUPRC and AUROC values of each method when applied to the subsets generated from each original dataset. Interestingly, we observed a roughly inverse relationship between the overall doublet-detection accuracy and the statistical stability. For example, DoubletFinder has the best overall accuracy in terms of both AUPRC and AUROC, yet its variation across data subsets is much greater than that of Scrublet, which has a much lower overall accuracy. Despite its suboptimal stability, we still found DoubletFinder as a top performer if we compare the lower-quartile accuracy (i.e., the 25-th percentile of AUPRC and AUROC values) of these methods. To summarize, even though statistical stability is an important criterion, in practice, it is often overruled by the overall accuracy reflected by the mean, median, or lower-quartile accuracy value. In terms of the overall accuracy, we found DoubletFinder, Solo, and hybrid as the top three methods.

Fourth, we evaluated the software implementation of doublet-detection methods, because user-friendliness, software quality, and active maintenance are crucial to the success of bioinformatics tools [72]. We scored each method in four aspects: software quality, execution convenience, publication, and documentation & support (Methods). Table 3 lists our score reasoning and the overall usability score of each method. In particular,DoubletDetection and DoubletDecon did not successfully run on one or more datasets. Regarding user support, Solo, DoubletDetection, DoubletFinder, and DoubletDecon have active Q&As on their software webpages for collecting users' feedback and answering users' questions. Among the nine methods, DoubletFinder achieved the highest usability score thanks to its excellent implementation.

2.3 Discussion

With the rapid development of scRNA-seq technologies, a skyrocketing number of computational methods have been developed for various scRNA-seq data analyses [16]. For example, since 2018, more than 45 imputation methods have already been developed to recover missing gene expression (commonly referred to as “dropouts”) in scRNA-seq data [24,28,29,31,73]. Such richness of computational methods is a double-sided blade. On the one hand, scRNA-seq researchers have more blocks to build analysis pipelines that accommodate their scientific investigation needs; on the other hand, it becomes increasingly difficult for researchers to choose the method, from dozens of methods developed for the same purpose, that best fits each step of their pipeline. Unlike in experimental sciences where new technologies often replace old ones, there are usually no clear-cut or universal choices of computational methods. An appropriate choice of computational method is case by case, depending on data characteristics and scientific questions at hand. Inappropriate method choices would, to varying extents, bias data analysis (such as by introducing artificial, non-biological signals) and ultimately lead to false discoveries [74,75]. To avoid this issue, the scRNA-seq field and the broad biomedical science community yearn for comprehensive benchmark studies that independently and fairly evaluate computational methods [24]. A well-designed benchmark study should offer users objective, accurate, and informative guidance on selecting the appropriate method(s) for a specific analysis task.

To provide the first, comprehensive benchmark of computational doublet-detection methods, in this study, we evaluated nine existing methods using 16 real and 112 synthetic scRNA-seq datasets from three perspectives: overall detection accuracy,

impacts on downstream analyses, and computational efficiency. We further categorized our benchmark results in nine aspects, including four related to doublet-detection accuracy and five associated with software implementation (Figure 5, which does not include DoubletDecon because it failed to run in most evaluations). In summary, DoubletFinder is the best method in terms of accuracy, yet its computational efficiency and stability are not among the best. The cxds method is the opposite: it has the best computational efficiency, excellent stability, but medium accuracy. Our summary is consistent with the aforementioned principle of computational methods that no method is universally the best, so a fair comparison of computational methods should be multifaceted.

Although our benchmark study has collected all the available scRNA-seq datasets to date that contain doublet annotations, we note that none of the annotations is utterly accurate due to experimental limitations. For example, the two species-mixture datasets, hm12k and hm6k, only labeled the heterotypic doublets formed by a human cell and a mouse cell; the six demuxlet datasets only labeled the doublets formed by cells of two individuals; many homotypic doublets were unlabeled in all these datasets. As a result, the incompleteness of doublet annotations would have inflated the false negative rates and reduced the precision of computational doublet-detection methods in our benchmark. To overcome this limitation, we designed extensive simulations to benchmark computational doublet-detection methods in a fair and comprehensive manner. Yet, how to generate accurate doublet annotations by experimental techniques remains an open question to experimental scientists.

Regarding the future development and benchmark of computational doublet-detection methods, here we list five open questions we deem important for computational scientists.

1. How to estimate the unknown doublet rate in a scRNA-seq dataset? Some methods provide heuristic guidance to estimate the doublet rates or select the threshold on doublet scores. For example, DoubletFinder suggests using the rates of heterotypic doublets and Poisson doublet formation as the respective lower and upper bounds of the expected doublet rate [41,50]; Scrublet recommends setting the doublet-score threshold in the middle of the two modes, which it expects to appear, in the doublet-score distribution [40]; Solo sets the doublet-score threshold to 0.5 by default [39]. However, there lacks consensus or direct estimation of the doublet rate from scRNA-seq data. To address this issue, we suggest estimating the null distribution of doublet scores (of singlets) as a preceding step; with a reliable null distribution estimate, estimating the doublet rate would then become feasible [76].
2. How to distinguish homotypic doublets from singlets? Existing computational doublet-detection methods cannot well identify the homotypic doublets that have similar transcriptome profiles to those of singlets, likely due to the ways they generate artificial doublets [39–41,45–48]. A possible direction is to extract and incorporate features that can distinguish homotypic doublets from singlets, such as the droplet library size.
3. How to distinguish doublets from droplets contaminated by ambient mRNA? Ambient mRNA molecules are released from lysed cells into the cell suspension; they may enter droplets and contaminate the measured transcriptome profiles of those droplets. Similar to doublets, contaminated droplets by ambient mRNA also

confound scRNA-seq data analysis [5]. Existing computational doublet-detection methods do not distinguish these two types of non-singlet droplets; instead, computational methods have been developed separately to detect contaminated droplets [77,78]. Ideally, the single-cell field desires a computational method that can simultaneously remove all non-singlet droplets, including doublets, contaminated droplets, and empty droplets, from scRNA-seq data.

4. How to improve doublet-detection algorithms regarding the use of artificial doublets? The majority of existing computational methods tackle the doublet detection task as a binary classification problem (Table 1). To train a classification algorithm, they use original droplets in data and artificial doublets they simulate to represent "singlets" and "doublets," respectively. However, not all original droplets are singlets, because otherwise we would not need doublet detection. By neglecting differences between original droplets and singlets, existing methods do not supply their classification algorithms with quality training data, and a likely consequence is that their post-training classifiers would be biased [79] and thus miss a substantial number of doublets among original droplets. A possible remedy for this drawback is to filter out the likely doublets from the original droplets, e.g., by applying outlier detection methods [80], before simulating artificial doublets and subsequently training a classification algorithm. An alternative remedy is to keep the training data but train a classification algorithm under the "learning with noise labels" machine-learning framework [79,81]. Moreover, there are possible improvements to be made in the generation of artificial doublets. Instead of simply adding or averaging the gene expression profiles of two random droplets as done in existing methods, finer

adjustments can be made to the mixing of two droplets so as to generate more realistic artificial doublets.

5. How to ensemble doublet-detection methods? As a multi-faceted problem, doublet detection can hardly be solved by one single computational method. This is due to the diversity of scRNA-seq datasets. The success of the method hybrid, an ensemble of two methods bcds and cxds, motivated us to think that ensembling reasonable and complementary methods, a technique widely used in machine learning [82,83], may boost the accuracy of doublet detection. Supplementary Tables S11 and S12 show the pairwise similarities of doublet-detection methods in terms of their doublet scores and identified doublets in the 16 real datasets. Seeing that the top-performing methods exhibited noticeable differences, we expect that there is room for using the ensemble technique to develop a more accurate doublet-detection method (see further discussion in the Supplementary).

By dissecting existing doublet-detection methods, we found method performance highly dependent on the values of hyperparameters (also known as tuning parameters), if any. For example, DoubletFinder, Scrublet, and doubletCells all use the *k*-nearest neighbor (kNN) algorithm to distinguish doublets from singlets; however, surprisingly, DoubletFinder outperformed the other two methods in most of our comparisons. A probable reason is that DoubletFinder optimizes several key hyperparameters of the kNN algorithm in a reasonable and data-driven way. For example, DoubletFinder selects the number of nearest neighbors *k* by maximizing the bimodality of the doublet score distribution. This advantage makes DoubletFinder adaptable to scRNA-seq datasets with distinct characteristics [40,41,48]. In contrast, Scrublet and doubletCells each assign a fixed

default value to *k*, restricting their flexibility and generalizability [40,41,48] (single-cell RNA sequencing discussion in the Supplementary). The choice of hyperparameter values is especially important for methods built upon complex algorithms. For example, bcds uses the gradient boosting algorithm [45], a leading classification algorithm that has more hyperparameters than the simple kNN algorithm does [84]; however, the additional complexity did not make bcds outperform DoubletFinder, probably due to the lack of hyperparameter optimization. This phenomenon emphasizes the importance for bioinformatics tools to optimize hyperparameter values in a scientific, data-driven way [85,86].

Ideally, doublet removal requires both experimental techniques and computational methods. If permitted, researchers may use an experimental technique and a computational method sequentially. That is, they first use an experimental technique such as multiplexing to filter out obvious doublets (e.g., the doublets formed by cells of different samples) and then apply a computational method to further screening for the remaining droplets that are likely doublets. Or they may combine the doublet scores assigned to each droplet by an experimental technique and a computational method, as proposed by the method Solo. This second approach requires the experimental technique to have a doublet scoring system [39].

In summary, computational doublet detection is critical for the quality control of scRNA-seq data analysis [5]. Our study is the first comprehensive benchmark of currently available doublet-detection methods under a wide variety of biological and technical settings. Our study provides much-needed guidance to researchers in choosing appropriate doublet-detection methods for scRNA-seq data analysis. Our results also

point out directions for further methodological development and improvement in computational doublet detection.

2.4 Methods

2.4.1 Real data preprocessing

Whenever preprocessed datasets were available, they were directly used in this study. Otherwise, datasets were preprocessed in the same way as in the original studies in which they were generated. In every dataset, genes and droplets were removed if they had no reads in any droplets and any genes, respectively. Below is the preprocessing detail for every dataset.

pbmc-ch [42]: human peripheral blood mononuclear cells (PBMCs) from eight donors. Doublets were annotated by cell hashing with CD45 as the hashing antibody. This dataset is available at https://www.dropbox.com/sh/ntc33ium7cg1za1/AAD_8XIDmu4F7lJ-5sp-rGFYa?dl=0 in files *pbmc_hto_mtx.rds* and *pbmc_umi_mtx.rds*. Its preprocessing pipeline is available at https://satijalab.org/seurat/v3.1/hashing_vignette.html, including an instruction about how to extract the doublet annotation.

cline-ch [42]: four human cell lines HEK, K562, KG1, and THP1. Doublets were annotated by cell hashing with CD29 and CD45 as the hashing antibodies. The access URL and preprocessing pipeline of this dataset are the same as those of the pbmc-ch dataset. The dataset is in files *hto12_hto_mtx.rds* and *hto12_umi_mtx.rds*.

Mkidney-ch [39]: dissociated mouse kidney cells. Doublets were annotated by cell hashing with cholesterol modified oligos (CMOs) as the hashing antibodies. The raw

count matrix and doublet annotations were downloaded from the Gene Expression Omnibus (GEO) [87] with the accession GSE140262.

hm-12k and **hm-6k** [52]: two mixtures of human HEK293T and mouse NIH3T3 cells with 12,000 and 6000 droplets respectively. The raw count matrices were downloaded from https://support.10xgenomics.com/single-cell-gene-expression/datasets/2.1.0/hgmm_12k and https://support.10xgenomics.com/single-cell-gene-expression/datasets/2.1.0/hgmm_6k.

A droplet was annotated as a doublet if its barcode was associated with both human and mouse. Mouse genes were mapped into their human orthologs using R package *biomaRt* [88] (v 2.44.1). Then each pair of human and mouse count matrices was concatenated into each of the two datasets.

pbmc-1A-dm, **pbmc-1B-dm**, and **pbmc-1C-dm** [43]: three samples of PBMCs from systemic lupus erythematosus (SLE) patients. Droplets were sequenced immediately after thawing. Doublets were annotated by demuxlet [43]. The raw count matrix and doublet annotations were downloaded from the GEO with the accession GSE96583.

pbmc-2ctrl-dm and **pbmc-2stiml-dm** [43]: two samples of PBMCs from SLE patients. Droplets were sequenced after being cultured for six hours following thawing, with (pbmc-2stiml-dm) or without (pbmc-2ctrl-dm) IFN-beta stimulation. Doublets were annotated by demuxlet. The raw count matrix and doublet annotations were downloaded from the GEO with the accession GSE96583.

J293t-dm [43]: a mixture of human Jurkat and HEK293T cell lines. Doublets were annotated by demuxlet. The raw count matrix was downloaded from https://ucsf.app.box.com/s/vg1bycvsjgyg63gkqsputprq5rxzjl6k/file/220975201845.

Doublet annotations were obtained from https://ucsf.app.box.com/s/vg1bycvsjgyg63gkqsputprq5rxzjl6k/file/220974993609.

pdx-MULTI [44]: a mixture of human breast cancer cells and mouse immune cells from a patient-derived xenograft (PDX) mouse model. Doublets were annotated by MULTI-seq [44]. The dataset was downloaded from the GEO with the accession GSE129578. Doublet were annotated by following the data processing pipeline available at https://github.com/chris-mcginnis-ucsf/MULTI-seq.

HMEC-orig-MULTI and **HMEC-rep-MULTI** [44]: human primary mammary epithelial cells (HMECs) with HMEC-orig-MULTI as the original sample and HMEC-rep-MULTI as a technical replica. The GEO accession and preprocessing pipeline of this dataset are the same as those of the pdx-MULTI dataset.

HEK-HMEC-MULTI [44]: a mixture of human HEK293Ts and HMECs. The GEO accession and preprocessing pipeline of this dataset are the same as those of the pdx-MULTI dataset.

nuc-MULTI [44]: a mixture of purified nuclei from human HEK293Ts, Jurkats, and mouse embryonic fibroblasts (MEFs). The GEO accession and preprocessing pipeline of this dataset are the same as those of the pdx-MULTI dataset. Mouse genes were mapped into their human orthologs using R package *biomaRt* (v 2.44.1).

2.4.2 Benchmark environment and parameter settings

All doublet-detection methods were executed on a server with two Intel(R) Xeon(R) E5-2687W v4 CPUs, 256GB memory, and Ubuntu 18.04 system. An Nvidia(R) Geforce(R) RTX 2080 Ti GPU was used to accelerate the execution of the Solo method as suggested [39]. The parameters of doublet-detection methods were set to their recommended values

or default values if no recommendation was available. The latest version of each method (by September 2020; Table 1) was used. Random seeds were fixed and saved in our code to ensure reproducibility. The detailed configuration for each method is summarized below.

doubletCells: The method was executed by following the instruction at https://bioconductor.statistik.tu-dortmund.de/packages/3.8/workflows/vignettes/simpleSingleCell/inst/doc/work-6-doublet.html. Doublet scores were obtained from the *doubletCells* function in R package *scran* (v 1.16.0) with parameters set to default.

Scrublet: R package *reticulate* (v 1.16) was used to execute the python module *scrublet* (v 0.2.1). The parameters were set by following the instruction at https://github.com/AllonKleinLab/scrublet/blob/master/examples/scrublet_basics.ipynb. Doublet scores were obtained from the function *Scrublet.scrub_doublets*.

cxds, **bcds** and **hybrid**: These three methods were executed by following the instructions at https://github.com/kostkalab/scds. Doublet scores were obtained from the functions *cxds, bcds* and *cxds_bcds_hybrid* in R package *scds* (v 1.2.0) with parameters set to default.

DoubletDetection: R package *reticulate* (v 1.16) was used to execute the python module *doubletdetection*. The parameters were set by following the instruction at https://nbviewer.jupyter.org/github/JonathanShor/DoubletDetection/blob/master/tests/notebooks/PBMC_8k_vignette.ipynb. The parameter *n_iters* was set to 5, as larger values were found to increase the running time significantly, but with little improvement in

performance. Doublet scores were obtained from the function *doubletdetection.BoostClassifier.fit*.

DoubletFinder: The method was executed by following the instruction at https://github.com/chris-mcginnis-ucsf/DoubletFinder. Doublet scores were obtained from the function *doubletFinder_v3* in R package *DoubletFinder* (2.0.3) with parameters set to default.

DoubletDecon: The method was executed by following the instruction at https://github.com/EDePasquale/DoubletDecon. Doublet predictions were obtained from the function *Main_Doublet_Decon* in R package *DoubletDecon* (v 1.1.5) with parameters set to default.

Solo: The method was executed by following the instruction at the GitHub repository https://github.com/calico/Solo. Every scRNA-seq count matrix was transformed into the *loom* format as required by the method. The parameters were set the same as those in the file *Solo_params_example.json*, which was downloaded from the GitHub repository. Doublet scores were obtained from the file *softmax_scores.npy*.

2.4.3 Measures of doublet-detection accuracy

Methodologically, computational doublet-detection methods employ binary classification algorithms to distinguish between two classes: singlets and doublets. AUPRC and AUROC, two measures of the overall accuracy of a binary classification algorithm, were used to evaluate the overall doublet-detection accuracy of each method. These two measures were calculated using the functions *pr.curve* and *roc.curve* in R package *PRROC* (v 1.3.1). Both functions input two vectors: the predicted doublet scores of true singlets and those of true doublets, and they output AUPRC and AUROC, one value each.

2.4.4 Simulation of scRNA-seq datasets containing doublets

All synthetic scRNA-seq datasets used in this study were generated in two steps. In Step 1, singlets in each dataset were generated by scDesign [51], which estimated a generative model of gene expression profiles from a real scRNA-seq dataset (cell type: HEK293t; protocol: 10x Genomics; gene number: 18760). The detailed experimental settings are described in the next subsection. In Step 2, given the number of singlets and a pre-specified doublet rate (i.e., the proportion of doublets among all droplets), the corresponding number of doublets were generated by random pairing of singlets. In detail, two randomly sampled singlets had their gene expression profiles (in UMI counts) averaged by gene, and that averaged profile is called a prototype doublet. For each of the 16 real scRNA-seq datasets, a doublet-to-singlet size ratio, defined as (average doublet library size)/(average singlet library size), was calculated. Then the library size of each prototype doublet was multiplied by a factor sampled from a normal distribution, whose mean and standard division were set to the mean and standard deviation of the 16 doublet-to-singlet size ratios. This scaling step turned prototype doublets into doublets, so that the doublet-to-singlet size ratios in the synthetic data were similar to those in the real data. Finally, the singlets used to generate doublets were removed. In mathematical terms, if X singlets were generated in Step 1 and the doublet rate was Y (a value between 0 and 1), then after Step 2 the numbers of doublets and singlets would be XY/(1+Y) and X(1-Y)/(1+Y), respectively, both rounded to the nearest integers. For example, if 1000 singlets were generated in Step 1 and the doublet rate was 20%, the numbers of doublets and singlets in the final dataset would be 167 and 667, respectively, making a total number of 834 droplets.

2.4.5 Experimental settings used in benchmarking simulations

80 scRNA-seq datasets were generated by scDesign to benchmark doublet-detection methods in four aspects: varying doublet rates, sequencing depths (i.e., per-cell library sizes), cell types, and between-cell-type heterogeneity levels.

- 20 synthetic datasets were generated with doublet rates increasing from 2% to 40% by a step size of 2%. The per-cell library size was set to 2000 UMI counts. All datasets contained two cell types. Based on the data generation scheme described in the last subsection, 500 singlets were generated for each cell type in Step 1. In Step 2, doublets were introduced based on each doublet rate, and the singlets used to generate doublets were removed.
- 20 synthetic datasets were generated with per-cell library sizes increasing from 500 to 10,000 UMI counts by a step size of 500 counts. All datasets contained two cell types. Based on the data generation scheme described in the last subsection, 500 singlets were generated for each cell type in Step 1. In Step 2, doublets were introduced based on a 20% doublet rate, and the singlets used to generate doublets were removed.
- 19 synthetic datasets were generated with numbers of cell types increasing from 2 to 20 by a step size of 1. The per-cell library size was set to 2000 UMI counts. Based on the data generation scheme described in the last subsection, 500 singlets were generated for each cell type in Step 1. In Step 2, doublets were introduced based on a 20% doublet rate, and the singlets used to generate doublets were removed.

- 21 synthetic datasets were generated with varying heterogeneity levels between two cell types. The heterogeneity level was controlled by four parameters (pUp, pDown, fU, and fL) in scDesign. Specifically, pUp and pDown denote the proportions of up- and down-regulated genes, and fU and fL define the upper and lower bounds of fold changes in the expression levels of DE genes. The following parameter combinations were used to generate 21 heterogeneity levels:

  Level 1: pUp = 0.010, pDown = 0.010, fU = 1.0, and fL = 0.5;

  Level 2: pUp = 0.012, pDown = 0.012, fU = 1.2, and fL = 0.6;

  …

  Level 21: pUp = 0.050, pDown = 0.050, fU = 5.0, and fL = 2.5.

  At all heterogeneity levels, the per-cell library size was set to 2000 UMI counts. Based on the data generation scheme described in the last subsection, 500 singlets were generated for each cell type in Step 1. In Step 2, doublets were introduced based on a 20% doublet rate, and the singlets used to generate doublets were removed.

2.4.6 DE gene analysis

One synthetic scRNA-seq dataset was generated by scDesign to have two cell types. The per-cell library size was 10,000 UMI counts. The pUp and pDown parameters in scDesign were both set to 0.03, suggesting that a total of 6% of genes were DE between the two cell types (3% up-expressed and 3% down-expressed). The fU and fL parameters in scDesign (i.e., the upper and lower bound of fold changes for DE genes) were set to 3 and 1.5, respectively. Based on the data generation scheme described in the Subsection “Simulation of scRNA-seq datasets containing doublets,” 500 singlets were generated for

each cell type in Step 1. In Step 2, doublets were introduced based on the 40% doublet rate, and the singlets used to generate doublets were removed. Three DE methods—DESeq2 [55], MAST [56], and the Wilcoxon rank-sum test [57] implemented in the R package *Seurat* (v 3.1.5) [61,62]—were applied to this dataset ("contaminated dataset" containing both singlets and doublets), its clean version without doublets ("clean dataset" only containing singlets), and its post-doublet-detection version after each doublet-detection method was applied (the top 40% droplets that received the highest doublet scores were removed). After each DE method was applied to every dataset, genes whose Bonferroni-corrected p-values did not exceed 0.05 were identified as DE. Three accuracy measures—precision, recall, and TNR—were calculated for every set of identified DE genes. For each DE method, its accuracy on the contaminated dataset and the clean dataset were used as the negative and positive controls, respectively, for benchmarking its accuracy on the post-doublet-detection datasets (Figure 2b–2c).

2.4.7 Identification of highly variable genes

Three synthetic datasets were generated with 10%, 20%, and 40% doublet rates, respectively. The per-cell library size was set to 2000 UMI counts. All datasets contained two cell types. Based on the data generation scheme described in the Subsection "Simulation of scRNA-seq datasets containing doublets," 500 singlets were generated for each cell type in Step 1. In Step 2, doublets were introduced based on each doublet rate, and the singlets used to generate doublets were removed. To identify the highly variable genes (HVGs), we applied the function *FindVariableFeatures* in R package *Seurat* (v 3.1.5) with default parameters to the three datasets ("contaminated datasets" containing both singlets and doublets; one dataset per doublet rate), their clean versions without

doublets (“clean datasets” only containing singlets), and their post-doublet-detection version after each doublet-detection method was applied (the top 10%, 20%, or 40% droplets that received the highest doublet scores were removed, and the removal percentage was set to the doublet rate). We refer to the identified HVGs as contaminated HVGs, clean HVGs, and post-doublet-detection HVGs, respectively. The Jaccard index between two sets of HVGs was calculated by the function *simi* in R package *proxy* (v 0.4-24) (Figure 2d).

2.4.8 Cell clustering analysis

Three synthetic scRNA-seq datasets were generated by scDesign to have four, six, and eight cell types. The per-cell library size was 2000 UMI counts. Based on the data generation scheme described in the Subsection “Simulation of scRNA-seq datasets containing doublets,” 500 singlets were generated for each cell type in Step 1. In Step 2, doublets were introduced based on a 20% doublet rate, and the singlets used to generate doublets were removed. The heterogeneity between cell types was determined by the default pUp, pDown, fU, and fL parameters in scDesign. After each doublet-detection method was applied to each dataset, the top x% of droplets, which received the highest doublet scores (with the removal percentage x% ranging from 0% to 25% by a step size of 1%), were removed; then two clustering algorithms—Louvain clustering implemented in R package *Seurat* (v 3.1.5) and DBSCAN [64] implemented in R package *dbscan* (v 1.1-5)—were used to identify cell clusters. Finally, the numbers of cell clusters were compared with the numbers of cell types to evaluate the effectiveness of doublet removal (Figure 2e; Supplementary Figure S2a). Whenever the number of cell clusters matched the number of cell types, the proportion of singlets among the remaining droplets was

used to measure each doublet-detection method's capacity for removing homotypic doublets (Figure 2f; Supplementary Figure S2b). In the example of four cell types, if a doublet-detection method (given a clustering algorithm) correctly led to four cell clusters under six removal percentages, then a proportion of singlets was calculated for each of the 24 clusters (four clusters times six removal percentages), resulting in 24 proportions.

2.4.9 Cell trajectory inference

Two scRNA-seq datasets were generated by Splatter [67] to have cell trajectories. Both datasets contained 1000 genes. In Step 1 of the data generation scheme described in the Subsection "Simulation of scRNA-seq datasets containing doublets," the first dataset had 500 singlets following a bifurcating trajectory, whose two branches had 250 singlets each, and the second dataset had 1000 singlets from a conjunction of three sequential trajectories, two of which had 333 singlets and the other had 334 singlets. In Step 2 for both datasets, doublets were introduced based on a 20% doublet rate, and the singlets used to generate doublets were removed. Parameters in Splatter were set to default except for de.prob and de.facLoc, which were set to 0.5 and 0.2, respectively. Each dataset was expanded into a suite, including its original version ("contaminated dataset"), clean version without doublets ("clean dataset"), and its post-doublet-detection version after each doublet-detection method was applied (the top 20% droplets that received the highest doublet scores were removed). For the first suite of datasets, cell trajectories were constructed by Slingshot [68] based on the pipeline available at https://github.com/kstreet13/slingshot/blob/master/vignettes/vignette.Rmd. For the second suite of datasets, the minimum spanning tree (MST) algorithm implemented in R package *slingshot* (v 1.6.1) was used to construct cell trajectories. The trajectories

constructed from the contaminated dataset and the clean dataset were used as the negative and positive controls, respectively, for benchmarking the trajectories inferred from the post-doublet-detection datasets (Figure 3a–2b).

In the temporally DE genes analysis, a scRNA-seq dataset with a single trajectory was generated by following the Slingshot pipeline available at https://github.com/kstreet13/slingshot/blob/master/vignettes/vignette.Rmd. This dataset contained 750 genes, whose temporal expression dynamics were categorized into four types: 500 stable genes with unchanged mean expression levels, 100 activated genes with increasing mean expression levels, 100 deactivated genes with decreasing mean expression levels, and 50 transient genes with mean expression levels first increasing and then decreasing, along the trajectory. The genes of the latter three types were defined as temporally DE genes. The mean expression levels of all 750 genes were specified by following the Slingshot pipeline. The per-cell library sizes were sampled from a negative binomial distribution with mean 1875 and dispersion 4. In the generation of a singlet, the 750 gene expression levels were sampled from a multinomial distribution with the number of trials as the (randomly sampled) per-cell library size and the probability of success as the 750 genes' normalized mean expression levels (summing up to 1). Following this, 300 singlets were generated in Step 1 of the data generation scheme described in the Subsection "Simulation of scRNA-seq datasets containing doublets." In Step 2, doublets were introduced based on a 20% doublet rate, and the singlets used to generate doublets were removed. After data generation, the pseudotime of each droplet was inferred by Slingshot and TSCAN on this dataset ("contaminated data"), its clean version without doublets ("clean data"), and its post-doublet-detection version after each doublet-

detection method was applied (the top 20% droplets that received the highest doublet scores were removed). Then for each dataset, we regressed each gene's expression levels in all droplets on the inferred pseudotime of the same droplets by the general additive model (GAM), which was implemented in the R function *gam*, and obtained a p-value. As a result, the genes with Bonferroni-corrected p-values under 0.05 were identified as temporally DE genes. Three accuracy measures—precision, recall, and TNR—were calculated for every set of identified temporally DE genes. The accuracy on the contaminated data and the clean data were used as the negative and positive controls, respectively, for benchmarking the accuracy on the post-doublet-detection data obtained by each doublet-detection method (Figure 3c–2d).

2.4.10 Distributed computing

We used two real scRNA-seq datasets pbmc-ch and pbmc-2ctrl-dm to compare the performance of doublet-detection methods under distributed computing. These two datasets are relatively large in our real data collection, containing 15,272 and 13,913 droplets (Table 1). For each doublet-detection method, its accuracy (AUPRC and AUROC) on the original datasets were used as the baselines. Next, the original dataset was randomly split into two, four, six, eight, and ten equally-sized batches for distributed computing. For every number of batches, each doublet-detection method was executed on each batch separately, the resulting doublet scores were concatenated across batches, and AUPRC and AUROC were calculated for the concatenated doublet scores and compared with the baselines (Figure 4a–b).

2.4.11 Scalability, stability, and usability

25 synthetic scRNA-seq datasets with varying numbers of droplets were generated by scDesign to examine the scalability of doublet-detection methods. Specifically, the number of genes was fixed to 5000, and the number of droplets increased from 400 to 10,000, with a step size of 400. Each doublet-detection method was executed on the 25 datasets, and the relationship between its running time and the number of droplets was plotted in Figure 4e.

Two real datasets, pbmc-ch and pbmc-2ctrl-dm, were used to evaluate the stability of doublet-detection methods. From each dataset, 20 subsets were generated by randomly subsampling 90% of droplets and 90% of genes. Each doublet-detection method was executed on all these subsets, and its stability was shown by the distributions of the resulting AUPRC and AUROC across subsets (Figure 4f).

Four criteria were defined for doublet-detection methods' usability: software quality, execution convenience, publication, and documentation & support. The software quality criterion indicates whether a doublet-detection method can be executed on all real and synthetic datasets used in this study. The execution convenience criterion is related to the popularity of the computational platform required to run a method. Methods written in R and Python packages are preferred because of the popularity of these two languages. The publication criterion is regarding whether a doublet-detection method has been published in a peer-reviewed journal. The documentation & support criterion evaluates a method's user-support resources, such as open-source code, tutorials, and active Q&As. Each criterion has three levels: excellent, good, and fair, corresponding to a score of 2,

1, and 0, respectively. The final usability score of a method was defined as the sum of the method's scores in these four criteria.

2.5 Acknowledgements

This chapter is based on the joint work with Dr. Jingyi Jessica Li. We would like to thank Dr. Bo Li at University of Texas Southwestern Medical Center (https://www.lilab-utsw.org/research) for bringing our attention to the doublet detection problem. We also appreciate the comments and feedback from our group members in the Junction of Statistics and Biology at UCLA (http://jsb.ucla.edu).

2.6 Figures and Tables

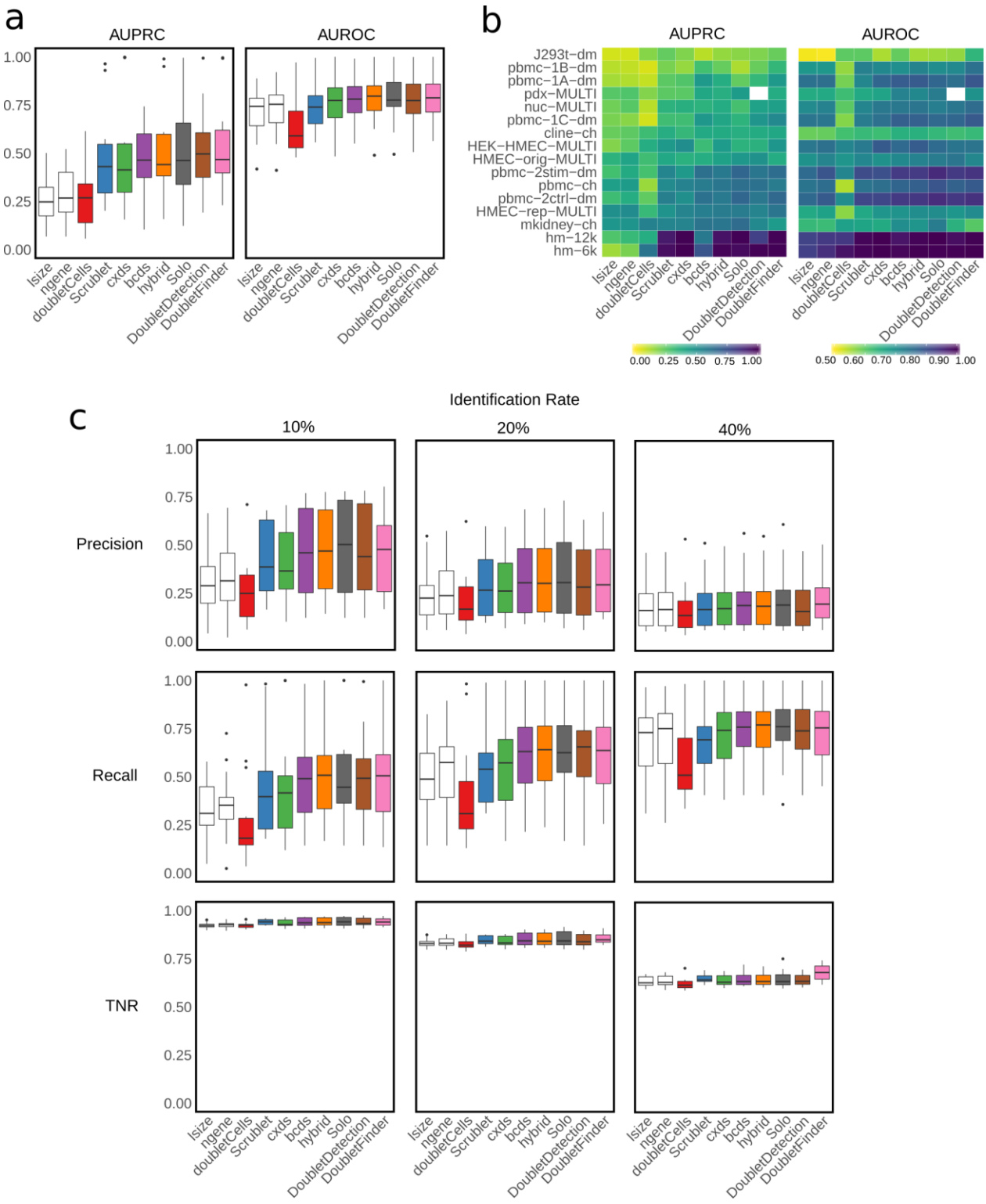


Figure 1. Evaluation of the eight doublet detection methods (except DoubletDecon) using 16 benchmark scRNA-seq datasets.

a-b, Performance (AUPRC and AUROC values) of each method applied to benchmark datasets, with (a) showing the distributions and (b) showing the values per dataset (white squares indicating failed runs); two baseline methods (lsize and ngene) are included in the comparison.

c, Precision, recall, and true negative rate (TNR) of each method under the 10%, 20%, or 40% identification rate, which is the percentage of droplets that received the highest doublet scores and were identified as doublets.

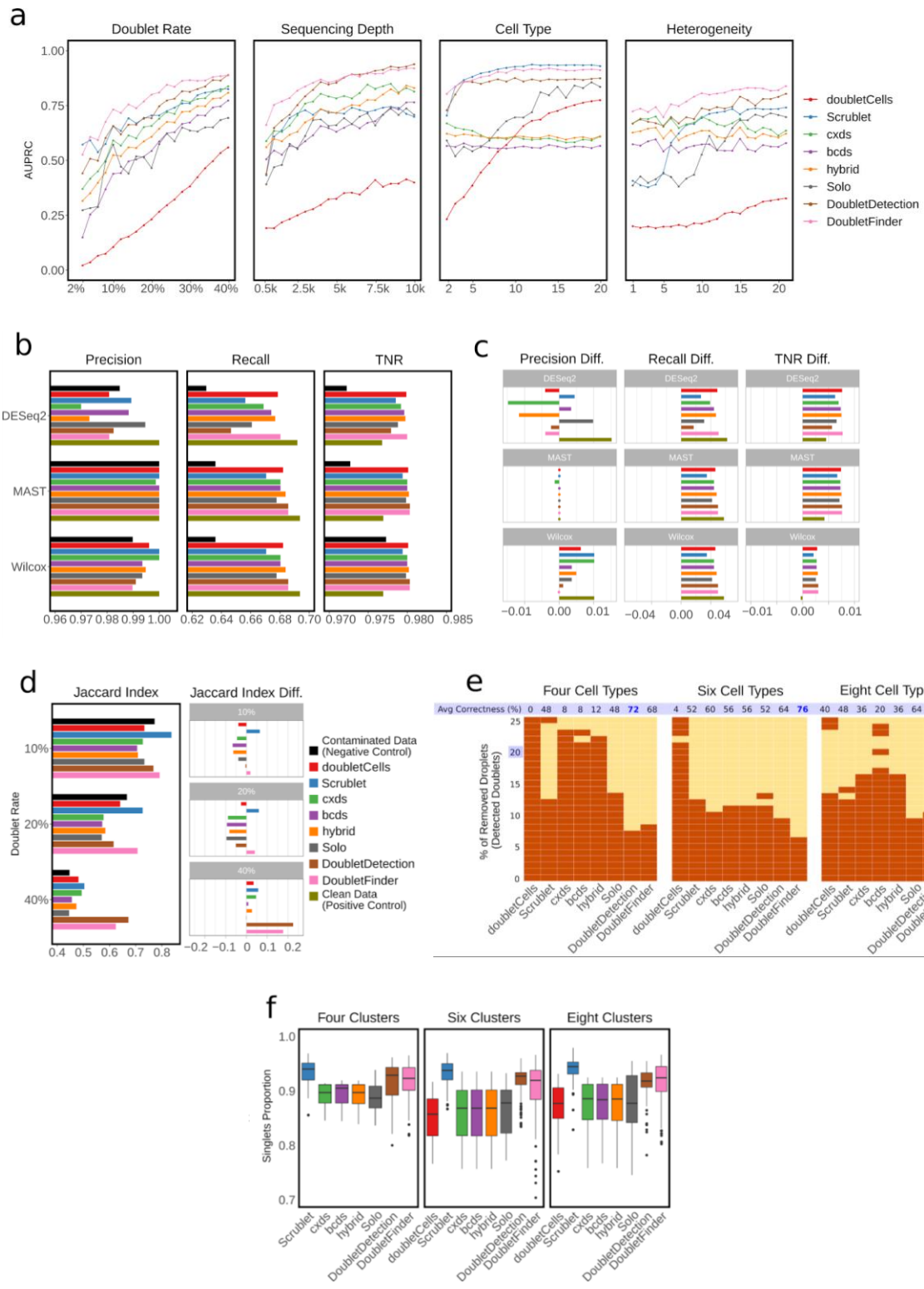

a
Doublet Rate
Sequencing Depth
Cell Type
Heterogeneity
AUPRC
doubletCells
Scrublet
cxds
bcds
hybrid
Solo
DoubletDetection
DoubletFinder
b
Precision
Recall
TNR
DESeq2
MAST
Wilcox
c
Precision Diff.
Recall Diff.
TNR Diff.
d
Jaccard Index
Jaccard Index Diff.
Doublet Rate
Contaminated Data (Negative Control)
Clean Data (Positive Control)
e
Four Cell Types
Six Cell Types
Eight Cell Types
Avg Correctness (%)
% of Removed Droplets (Detected Doublets)
Correct
Incorrect
f
Four Clusters
Six Clusters
Eight Clusters
Singlets Proportion

Figure 2. Evaluation of the eight doublet detection methods (except DoubletDecon) using four simulation studies, and the effects of doublet detection on DE analysis, highly variable genes (HVG) identification, and cell clustering.

a, Performance (AUPRC values) of each method in four simulation settings: varying doublet rates (from 2% to 40% with a step size of 2%), varying sequencing depths (from 500 to 10,000 UMI counts per cell, with a step size of 500 counts), varying numbers of cell types (from 2 to 20 with a step size of 1), and 20 heterogeneity levels, which specify the extent to which genes are differentiated between two cell types (Methods).

b, Precision, recall, and TNR by each of three differential expression (DE) methods: DESeq2, MAST, and the Wilcoxon rank-sum test (Wilcox), after each of the eight doublet detection methods was applied to a simulated dataset; for negative and positive controls, we included the DE accuracies on the contaminated data with 40% doublets and the clean data without doublets.

c, We re-illustrate the results in b) by showing the improved DE accuracy in each metric (precision, recall, and TNR) after removing detected doublets from the contaminated data; the results on the clean data without doublets are shown as a positive control.

d, Left panel: the Jaccard index between the post-doublet-detection HVGs of each doublet-detection method and the clean HVGs under the 10%, 20%, or 40% doublet rate. The Jaccard index between the contaminated HVGs and the clean HVGs was used as negative control for each doublet rate. Right panel: illustration of the left panel; the improved Jaccard indices upon the negative controls (i.e., Jaccard index differences) after the detected doublets by each method were removed from the contaminated data.

e, Cell clustering result by the Louvain algorithm after each of the eight doublet-detection method was applied to remove a varying percentage of droplets as the identified doublets (y-axis, from 0% to 25% with step size of 1%); the true numbers of cell clusters are four, six, and eight under three simulation settings, each containing 20% true doublets; the yellow color indicates that the correct number of clusters was identified, while the red color indicates otherwise. The true percentage of doublets, 20%, is highlighted in blue. For each method, its average correctness (i.e., the percent of yellow colors across all the removal percentages) is also highlighted in blue.

f, Under the same three simulation settings as in a), the distributions of the singlet proportions are shown after doublet removal by each method, if the remaining droplets led to the correct number of cell clusters in a); doubletCells is not shown for the four-cluster setting because it did not lead to the correct number of cell clusters in a).

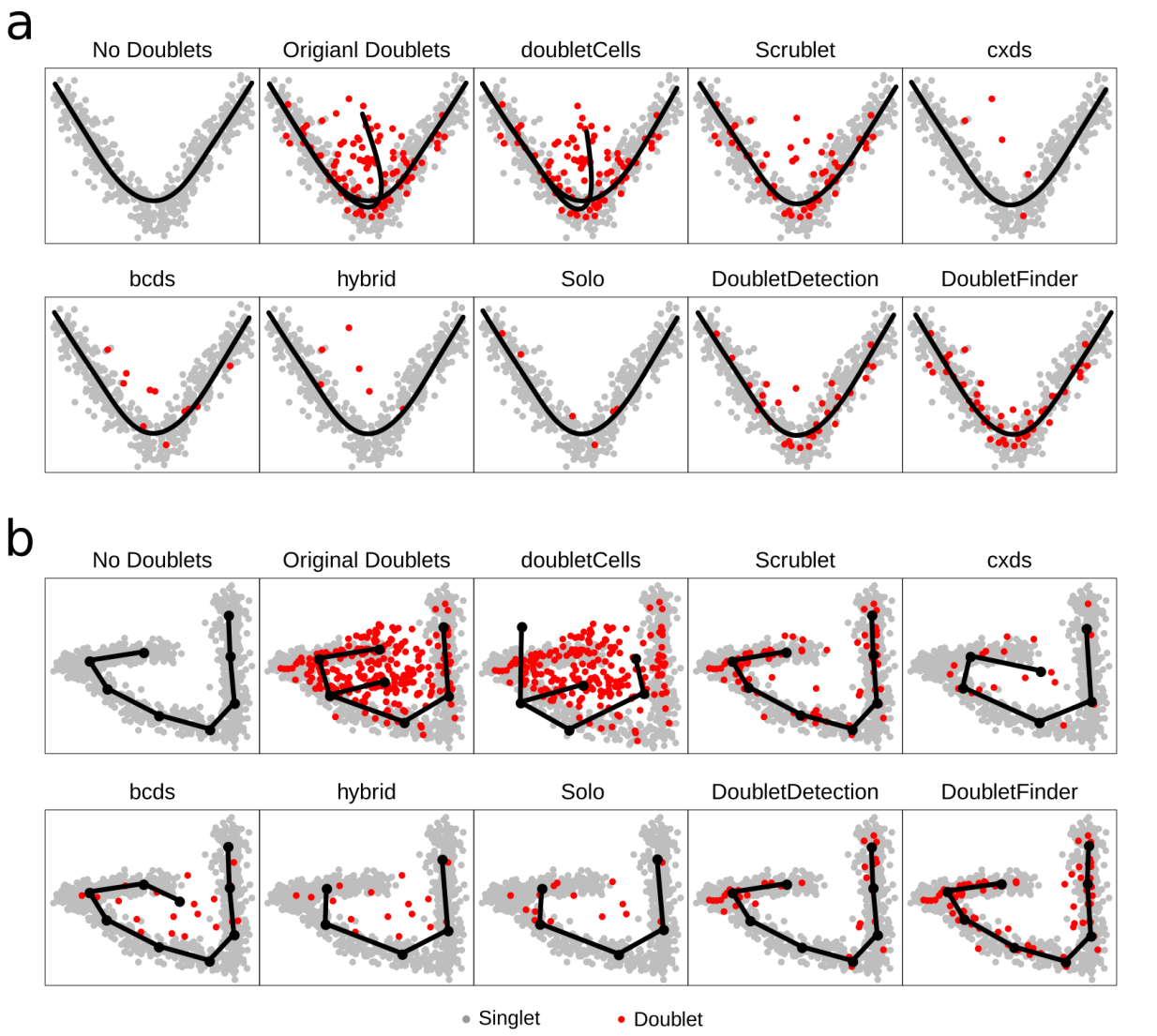

a
No Doublets
Origianl Doublets
doubletCells
Scrublet
cxds
bcds
hybrid
Solo
DoubletDetection
DoubletFinder
b
No Doublets
Original Doublets
doubletCells
Scrublet
cxds
bcds
hybrid
Solo
DoubletDetection
DoubletFinder
Singlet
Doublet
c
Slingshot
TSCAN
precision
recall
TNR
0.5 0.6 0.7 0.8 0.9 1.0
0.25 0.50 0.75 1.00
Original Doublets
doubletCells
Scrublet
cxds
bcds
hybrid
Solo
DoubletDetection
DoubletFinder
No Doublets
d
Slingshot Diff.
TSCAN Diff.
precision
recall
TNR
−0.2 0.0 0.2

Figure 3. Effects of doublet detection on cell trajectory inference.

a, Trajectories constructed by Slingshot after each of the eight doublet-detection methods was applied to remove the identified doublets, whose percentage among all the droplets was set to 20%, the percentage of true doublets in the simulated dataset. The true cell topology is bifurcating. For negative and positive controls, we included the trajectories constructed on the original dataset with 20% doublets and its cleaned version without doublets.

b, Trajectories constructed by minimum spanning tree (MST) after each of the eight doublet detection methods was applied to remove the identified doublets, whose percentage among all the droplets was set to 20%, the percentage of true doublets in the simulated dataset. The true cell topology is a conjunction of three trajectories. For negative and positive controls, we included the trajectories constructed on the original dataset with 20% doublets and its cleaned version without doublets.

c, Precision, recall, and TNR of temporally differentially expressed genes identified by the general additive model (GAM) applied to trajectories constructed by Slingshot and TSCAN, after each of the eight doublet-detection method was applied to remove the identified doublets, whose percentage among all the droplets was set to 20%, the percentage of true doublets in the simulated dataset. The true cell topology is a single lineage. For negative and positive controls, we included the accuracy of temporally differentially expressed genes identified from the contaminated data with 20% doublets and the clean data without doublets.

d, We re-illustrate the results in c) by showing the improved accuracy in each metric (precision, recall, and TNR) after removing detected doublets from the contaminated data; the results on the clean data without doublets are shown as a positive control.

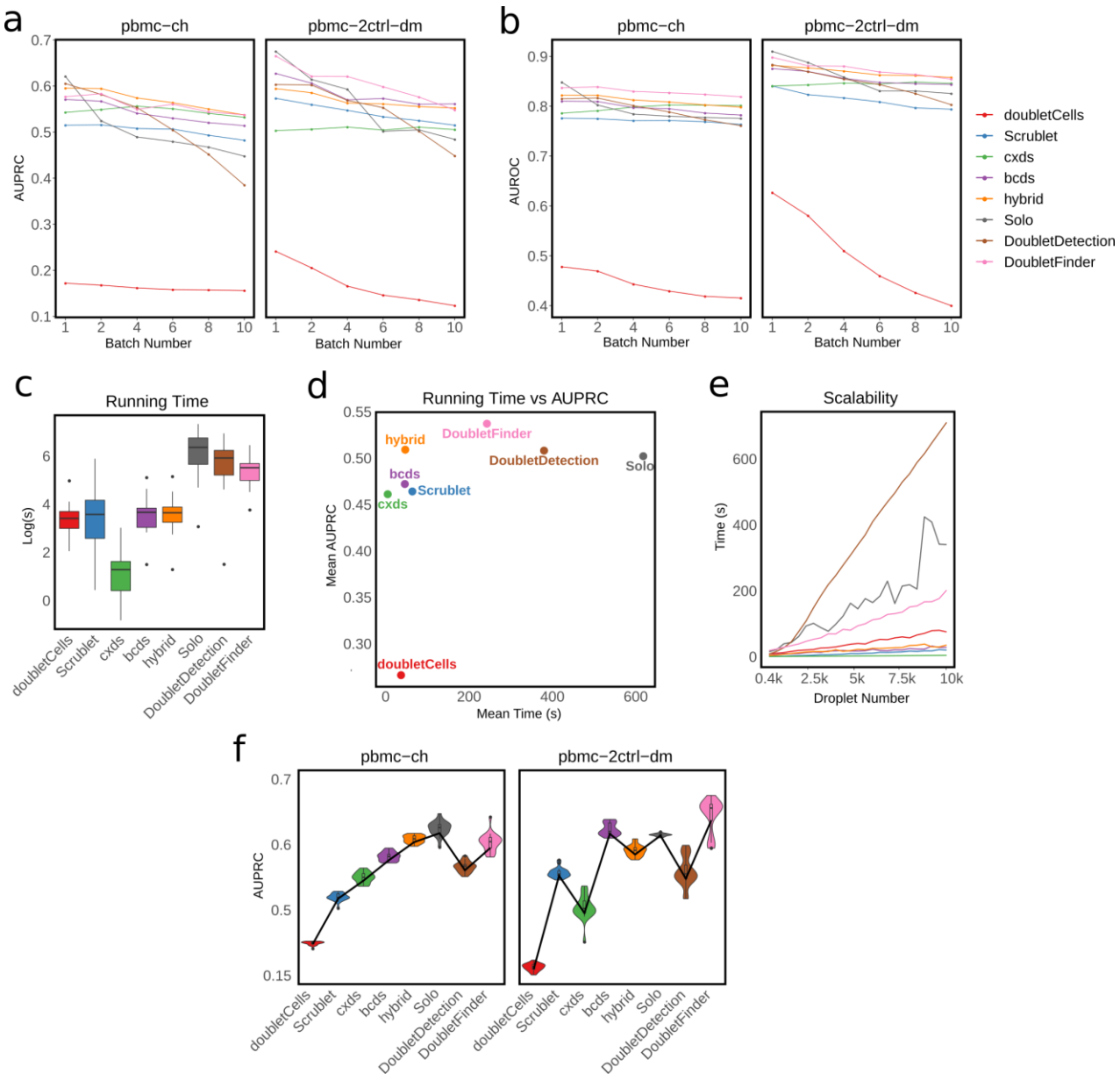


Figure 4. Comparison of doublet detection methods in terms of distributed computing, running time, scalability, and stability.

a-b, Distributed computing performance of each method on two real datasets pbmc-ch and pmc-2ctrl-dm. We first divided the original datasets into varying numbers of batches with equal sizes; then we applied each method to individual batches separately to identify and remove doublets; finally we pooled batches together to assess the detection accuracy (AUPRC and AUROC values) of each method.

c, Distribution of running time in (natural log) seconds of each method across 16 real datasets.

d, Mean AUPRC vs. mean running time (across 16 real datasets) of eight doublet-detection methods.
e, Scalability of each method. We calculated the relationship between running time and droplet number for each method on simulated datasets with varying droplet numbers.
f, Stability of each method. We generated 20 datasets by randomly subsampling 90% droplets and 90% genes from the real datasets pbmc-ch and pbmc-2ctrl-dm, and we applied each method to all the subsampled datasets. For each real dataset, the distribution of AUPRC values of each method across subsampling is shown, with 25% quantiles connected. We use the variance of the distribution to measure the stability of each method.

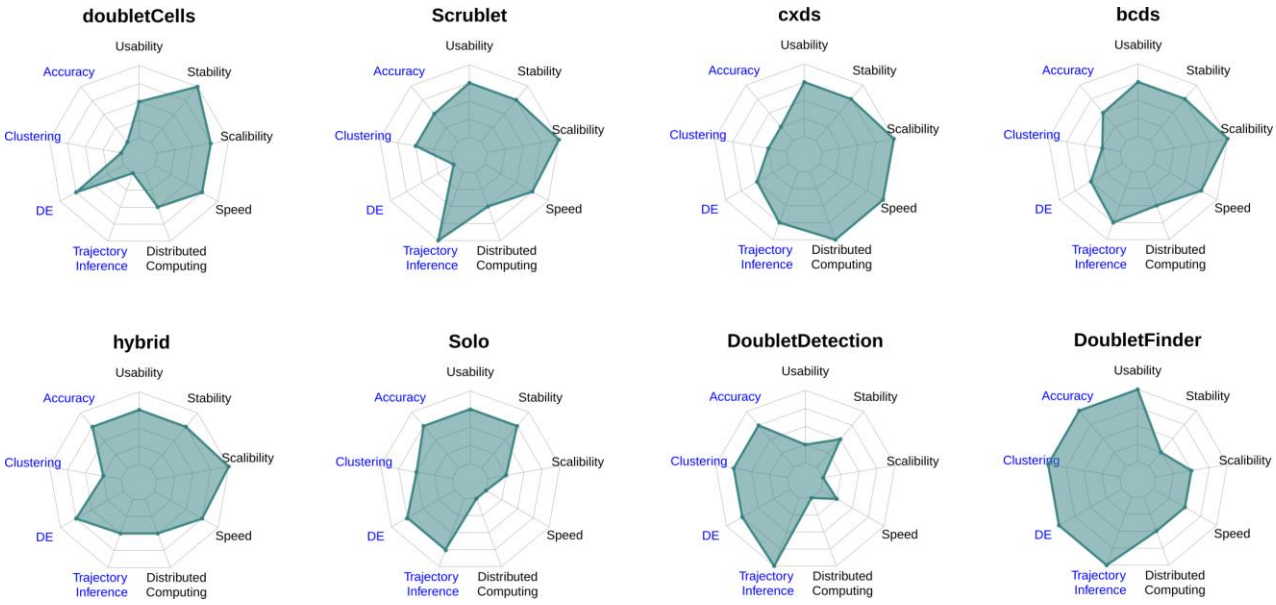


Figure 5. A graphical summary of benchmark results. The four aspects related to doublet detection accuracy are marked in blue, while other five aspects related to software implementation are marked in black.

Table 1. An overview of nine computational doublet-detection methods evaluated in this study.

Method	Programming language	Artificial doublets	Dimension reduction	Algorithm description
Scrublet [40]	Python	Yes	Principal component analysis (PCA)	It generates artificial doublets by adding two randomly selected droplets' gene expression profiles. The doublet score of each droplet is defined as the proportion of artificial doublets among its *k*-nearest neighboring droplets in the principal component (PC) space, whose number of dimensions is specified by users.
doubletCells [48]	R	Yes	PCA	It generates artificial doublets by adding two randomly selected droplets' gene expression profiles. For each droplet, it calculates the proportion of artificial doublets, p_A , in a neighborhood in the PC space, whose number of dimensions is specified by users. The radius of the neighborhood is set to be the median distance from the droplet to its 50th nearest neighbor. The doublet score of each droplet is defined as $p_A/(1-p_A)^2$.
cxds [45]	R	No	Highly variable genes	It calculates a p-value for each pair of genes under the null hypothesis that the number of droplets where exactly one of the two genes is expressed follows a binomial distribution. The doublet score of each droplet is defined as the sum of negative (natural) log p-values of co-expressed gene pairs, where two genes in each pair both have non-zero expression levels in this droplet.
bcds [45]	R	Yes	Highly variable genes	It generates artificial doublets by adding two randomly selected droplets' gene expression profiles and pools these artificial doublets with the original droplets. Then it trains a gradient boosting classifier to classify the pooled droplets into original droplets and artificial doublets. The doublet score of each droplet is defined as the predicted probability of being an artificial doublet.
hybrid [45]	R	-	-	It normalizes the doublet scores of cxds and bcds to values between 0 and 1. The doublet score of each droplet is defined as the sum of the two normalized doublet scores.
DoubletDetection [47]	Python	Yes	PCA	It generates artificial doublets by adding two randomly selected droplets' gene expression profiles and pools these artificial doublets with the original droplets. Then it conducts Louvain clustering on the pooled droplets. For each droplet cluster, it performs a hypergeometric test and computes p-value = 1 - hypergeom.cdf(N, K, n, k), where N is the number of droplets, K is the number of artificial doublets, n is the number of droplets in this cluster, and k is the number of artificial doublets in this cluster. All droplets in this cluster will have the same p-value. It repeats the above steps (starting from artificial doublet generation) for a user-specified number of runs. The doublet score of each droplet is defined as its average p-value across all runs.
DoubletFinder [41]	R	Yes	PCA	It generates artificial doublets by averaging two randomly selected droplets' gene expression profiles. The doublet score of each droplet is defined as the proportion of artificial doublets among its *k*-nearest neighboring droplets in the principal component (PC) space, whose number of dimensions is specified by users. The number of neighbors, k, is selected by maximizing the mean-variance normalized bimodality coefficient [89] of the distribution of doublet scores.

Solo [39]	Linux command	Yes	Variational autoencoder	For a randomly selected droplet pair, it estimates a multinomial distribution whose number of trials equals the sum of total counts in these two droplets and whose event probabilities equal the gene proportions calculated from the mean gene expression profile of these two droplets. Then it generates artificial doublets by randomly sampling a gene expression profile from this multinomial distribution. That is, the number of artificial doublets equals the number of randomly selected droplet pairs. These artificial doublets are pooled with the original droplets. Then it trains a neural network to classify the pooled droplets into original droplets and artificial doublets. The doublet score of each droplet is defined as the predicted probability of being an artificial doublet.
DoubletDecon [46]	R	Yes	Deconvolution	It generates artificial doublets by taking a weighted average of two randomly selected droplets' gene expression profiles (the default weights are 0.7 and 0.3). Putative doublets are defined as those droplets whose gene expression profiles after deconvolution [90] are concentrated on the centroids of artificial doublet clusters. Finally, it defines doublets as those putative doublets whose gene expression profiles are dissimilar to those of original droplet clusters.

Table 2. 16 real scRNA-seq datasets with experimentally annotated doublets used in this study.

Dataset	Doublet annotation technique	Cell types	Droplet #	Gene #	Doublet rate	Median UMI count	Median # of expressed genes	Reference
pbmc-ch	Cell hashing	pbmc	15272	21639	16.66%	556	323	[42]
cline-ch	Cell hashing	HEK293T, K562, KG1, THP1	7954	25221	18.42%	4824	2149	
mkidney-ch	Cell hashing	Mouse kidney	21179	18940	37.31%	3929	1687	[39]
hm-12k	Species mixture	HEK293T, NIH3T3	12820	15106	5.69%	12424	3147	[52]
hm-6k	Species mixture	HEK293T, NIH3T3	6806	15080	2.51%	21301	4032	
pbmc-1A-dm	demuxlet	pbmc	3298	15170	3.64%	973	384	[43]
pbmc-1B-dm	demuxlet	pbmc	3790	15143	3.43%	862	361	
pbmc-1C-dm	demuxlet	pbmc	5270	15865	6.00%	829	352	
pbmc-2ctrl-dm	demuxlet	pbmc	13913	17584	11.49%	1276	526	
pbmc-2stim-dm	demuxlet	pbmc	13916	17315	11.72%	1360	550	
J293t-dm	demuxlet	Jurkat, HEK293T	500	16374	8.40%	14134	3461	
pdx-MULTI	MULTI-seq	Human breast cancer, mouse immune	10296	14025	12.79%	2242	1029	[44]
HMEC-orig-MULTI	MULTI-seq	HMEC	26426	24199	13.50%	23502	4598	
HMEC-rep-MULTI	MULTI-seq	HMEC	10580	17473	31.02%	1188	601	
HEK-HMEC-MULTI	MULTI-seq	HEK293T, HMEC	10641	23982	4.60%	17424	3795	
nuc-MULTI	MULTI-seq	nuclei (HEK293T, MEF, Jurkat)	5578	21490	8.52%	1021	786	

Table 3. Usability of the nine doublet-detection methods. We measured the usability of each method in four aspects: software quality, execution convenience, publication, and documentation & support. Each aspect has three levels: excellent, good, and fair, which correspond to scores 2, 1, and 0, respectively. The usability score of a method is the sum of its four scores under the four aspects.

<table>
<tr><th></th><th>Software quality</th><th>Execution convenience</th><th>Publication</th><th>Documentation & support</th><th>Usability score</th></tr>
<tr><td>doubletCells</td><td rowspan="5">Excellent
(success on all datasets)</td><td>Excellent
(R package)</td><td>Good
(published as a part of a research paper in peer-reviewed journal)</td><td>Good
(documentation, custom webpage, but no Q&A)</td><td>6</td></tr>
<tr><td>Scrublet</td><td>Excellent
(Python module)</td><td rowspan="4">Excellent
(published as an independent research paper in a peer-reviewed journal)</td><td rowspan="4">Good
(documentation, GitHub webpage, but no Q&A)</td><td>7</td></tr>
<tr><td>cxds</td><td rowspan="3">Excellent
(R package)</td><td>7</td></tr>
<tr><td>bcds</td><td>7</td></tr>
<tr><td>hybrid</td><td>7</td></tr>
<tr><td>Solo</td><td>Good
(Linux command-line with a stringent requirement on input data format: loom/hd5)</td><td>Excellent
(published as an independent research paper in a peer-reviewed journal)</td><td rowspan="3">Excellent
(documentation, GitHub webpage, and active Q&A)</td><td>7</td></tr>
<tr><td>DoubletDetection</td><td>Good
(failure on one real dataset)</td><td>Excellent
(Python module)</td><td>Fair
(GitHub webpage, manuscript with algorithm description)</td><td>5</td></tr>
<tr><td>DoubletFinder</td><td>Excellent
(success on all datasets)</td><td>Excellent
(R package)</td><td>Excellent
(published as an independent research paper in a peer-reviewed journal)</td><td><u>8</u></td></tr>
<tr><td>DoubletDecon</td><td>Fair
(failure on four real datasets and the majority of synthetic datasets)</td><td>Excellent
(R package)</td><td>Excellent
(published as an independent research paper in a peer-reviewed journal)</td><td>Excellent
(documentation, GitHub webpage, and active Q&A)</td><td>6</td></tr>
</table>

2.7 Supplementary Materials

2.7.1 Accuracy of computational doublet detection in relation to experimental techniques for doublet labeling

Four experimental techniques were used to label doublets in the 16 real datasets used in this study: cell hashing [42], species mixture [40], demuxlet [43], and MULTI-seq [44]. To examine the relationship between the accuracy of computational doublet-detection methods and the use of experimental techniques for doublet labeling, we calculated the mean AUPRC of each computational method across the datasets labeled by each experimental technique (Supplementary Figure S2d; Supplementary Table S10). Overall, all computational doublet-detection methods achieved the highest accuracy on the species-mixture datasets, followed by the cell-hashing, MULTI-seq, and demuxlet datasets. This is an expected result since doublet-detection methods are more capable of identifying heterotypic doublets than homotypic doublets by design [39–41,45–48], and all the labeled doublets in the species-mixture datasets are heterotypic (i.e., formed by cells of two species); meanwhile, the cell-hashing, MULTI-seq, and demuxlet datasets contain labeled doublets that are both heterotypic and homotypic (e.g., formed by cells of the same type from two samples or individuals), and they miss certain heterotypic doublets (e.g., formed by cells of different types from the same sample or individual). Among the eight doublet-detection methods (excluding DoubletDecon which cannot generate doublet scores), DoubletFinder, cxds, and Solo achieved the highest detection accuracy on the species-mixture datasets, demonstrating their strength of identifying heterotypic doublets. DoubletFinder was also the top performer on the MULTI-seq and demuxlet

datasets in terms of mean AUPRC, while Solo excelled on the cell-hashing datasets. Interestingly, cxds exhibited the largest performance discrepancy between the species-mixture datasets and the other three types of datasets, highlighting its stronger priority towards identifying heterotypic doublets than other methods'.

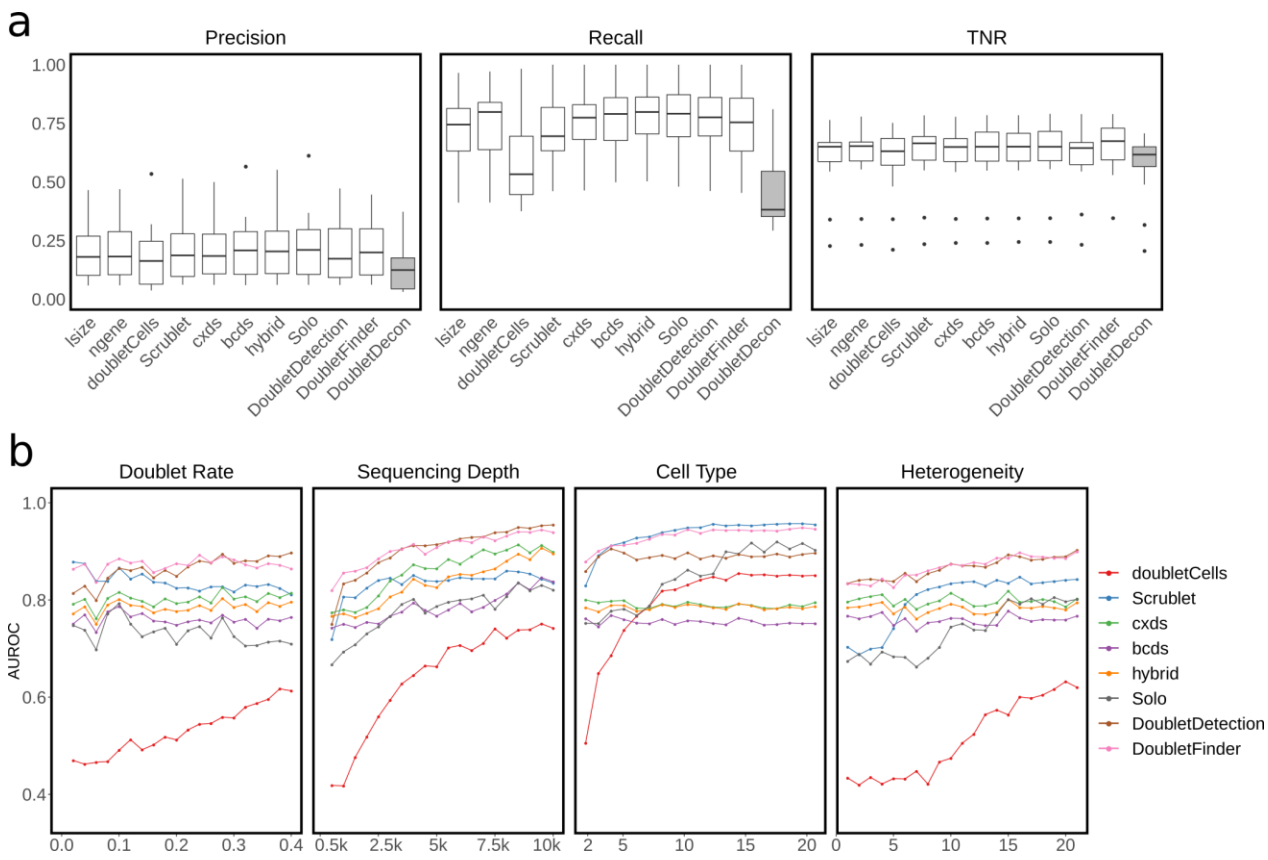
2.7.2 Pairwise similarities of computational doublet-detection methods

First, we calculated the Pearson correlation coefficient between every two doublet-detection methods (except hybrid, which is an ensemble of bcds and cxds, and DoubletDecon, which cannot generate doublet scores) in terms of their doublet scores in each of the 16 benchmark datasets; for every pair of methods, we averaged their 16 Pearson correlation coefficients (Supplementary Table S11). Among the 21 pairs of methods, DoubletFinder-DoubletDetection, Solo-bcds, and DoubletFinder-bcds have the largest mean correlations. Second, we calculated the Jaccard index between every two doublet-detection methods (except hybrid and DoubletDecon) in terms of their identified doublets, whose numbers are set equal to the number of labeled doublets, in each of the 16 benchmark datasets; for every pair of methods, we averaged their 16 Jaccard indices (Supplementary Table S12). Among the 21 pairs of methods, DoubletFinder-DoubletDetection, DoubletDetection-Solo, and DoubletFinder-Solo have the largest mean Jaccard indices, which reflect the large overlaps of their identified doublets. These two similarity analyses indicate the possibility of developing an ensemble method to combine the top-performing methods that are not too similar [83]. Given the high accuracy of DoubletFinder and the distinctive algorithm design of cxds (the only method without artificial doublets), these two methods may serve as good candidates to be combined into an ensemble method.

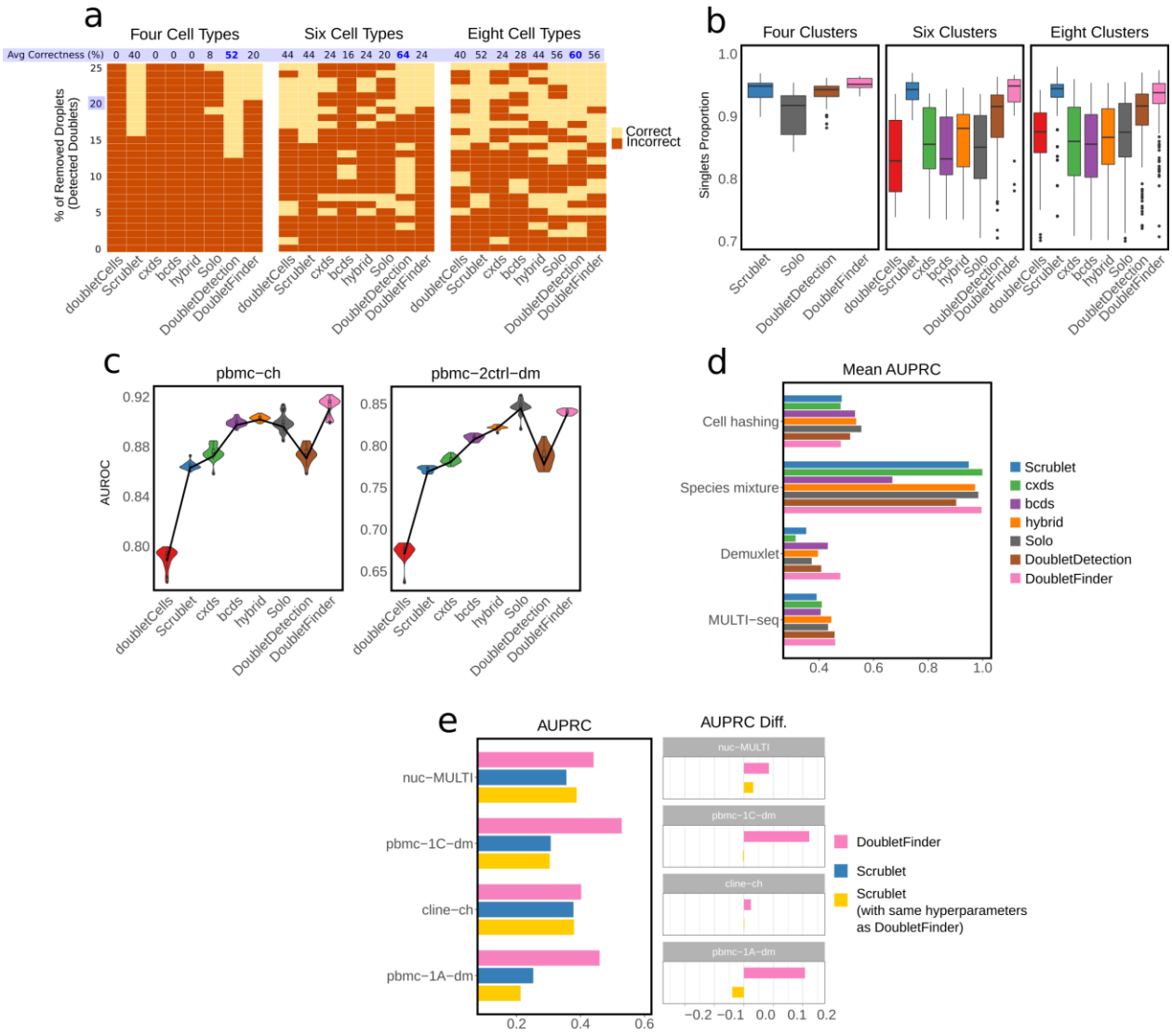
2.7.3 Comparison of hyperparameter selection in knn-based methods

The algorithm designs of Scrublet and DoubletFinder are similar because they both define each droplet's doublet score as the proportion of artificial doublets among the *k*-nearest neighbors of this droplet in the principal component (PC) space. The major difference between Scrublet and DoubletFinder is how they select hyperparameters, including the number of artificial doublets to generate, the number of genes used to perform the principal component analysis, the number of PCs to define nearest neighbors, and the number of nearest neighbors *k*. Supplementary Table S13 summarizes the default hyperparameter settings of Scrublet and DoubletFinder. In particular, DoubletFinder automatically selects *k* by maximizing the mean-variance normalized bimodality coefficient [89] of the distribution of doublet scores. To examine the effect of hyperparameter selection on the method performance, we selected four real datasets on which DoubletFinder outperformed Scrublet, and replaced the hyperparameters of Scrublet by those of DoubletFinder, including the *k*s selected by DoubletFinder for those datasets. Supplementary Figure S2e summarizes the AUPRC values of three methods—DoubletFinder, Scrublet with default hyperparameters, and Scrublet with the same hyperparameters as DoubletFinder—on each of the four datasets. With the hyperparameters of DoubletFinder, Scrublet improved its detection accuracy on two datasets, nuc-MULTI and pbmc-1C-dm, but it still underperformed DoubletFinder. On the other two datasets, cline-ch and pbmc-1A-dim, Scrublet performed similarly or even worse, respectively, with the hyperparameters of DoubletFinder. This result suggests that hyperparameter selection is an important but not the only factor that determines the

performance of doublet-detection methods. Other aspects of algorithm design, including the generation of artificial doublets and algorithm implementation, also play critical rules.



Supplementary Figure S1. **a,** Comparison between DoubletDecon (grey) and other methods in terms of precision, recall, and true negative rates (TNRs) on 16 benchmark scRNA-seq datasets. The number of doublets is determined by the prediction result of DoubletDecon. Two baseline detection methods (lsize and ngenes) are included in the comparison. **b,** Performance (AUROC values) of each method in four simulation settings: varying doublet rates (from 2% to 40% with a step size of 2%), varying sequencing depth (from 500 to 10,000 UMI counts per cell, with a step size of 500 counts), varying numbers of cell types (from 2 to 20 with a step size of 1), and 20 heterogeneity levels, which specify the extent to which genes are differentiated between two cell types (see Methods).



Supplementary Figure S2. **a,** Cell clustering result by the DBSCAN algorithm after each of the eight doublet-detection method was applied to remove a varying percentage of droplets as the identified doublets (y-axis, from 0% to 25% with step size of 1%); the true numbers of cell clusters are four, six, and eight under three simulation settings, each containing 20% true doublets; the yellow color indicates that the correct number of clusters was identified, while the red color indicates otherwise. The true percentage of doublets, 20%, is highlighted in blue. For each method, its average correctness (i.e., the percent of yellow colors across all the removal percentages) is also highlighted in blue. **b,** Under the same three simulation settings as in a), the distributions of the singlet proportions are shown after doublet removal by each method, if the remaining droplets led to the correct number of cell clusters in a); doubletCells, cxds, bcds, and hybrid are not shown for the four-cluster setting because it did not lead to the correct number of cell clusters in a). **c,** Stability of each method. We generated 20 datasets by randomly

subsampling 90% cells and 90% genes from the real datasets pbmc-ch and pbmc-2ctrl-dm, and we applied each method to all the subsampled datasets. For each real dataset, the distribution of AUPRC values of each method across subsampling is shown, with 25% quantiles connected. We use the variance of the distribution to measure the stability of each method. **d,** Mean AUPRC of each doublet-detection method across the real datasets with doublets labeled by each of four experimental techniques (cell hashing, species mixture, demuxlet, and MULTI-seq). Due to the low mean AUPRC values of doubletCells, we excluded it to show a more clear comparison of the other methods. The mean AUPRC of doubletCells can be found in Supplementary Table S10. **e,** AUPRCs of DoubletFinder, Scrublet with default hyperparameters, and Scrublet with same hyperparameters as DoubletFinder on four real datasets (nuc-MULTI, pbmc-1C-dm, cline-ch, and pbmc-1A-dm).

Supplementary Table S1. AUPRC values of ten doublet-detection methods, including two baselines lsize and ngene, applied to 16 benchmark scRNA-seq datasets. The top-performing method on each dataset is boldfaced and underlined.

	lsize	ngene	doubletCells	Scrublet	cxds	bcds	hybrid	Solo	DoubletDetection	DoubletFinder
pbmc-ch	0.438	0.449	0.150	0.526	0.556	0.583	0.609	**0.641**	0.624	0.584
cline-ch	0.231	0.246	0.311	0.378	0.332	0.396	0.391	0.372	0.389	**0.402**
mkidney-ch	0.476	0.483	0.565	0.546	0.549	0.618	0.607	**0.651**	0.529	0.454
hm-12k	0.274	0.326	0.382	0.932	**0.998**	0.594	0.952	0.995	0.810	0.994
hm-6k	0.142	0.200	0.615	0.965	**1.000**	0.743	0.991	0.972	0.995	0.997
pbmc-1A-dm	0.134	0.115	0.088	0.252	0.273	0.458	0.381	0.239	0.333	**0.460**
pbmc-1B-dm	0.109	0.092	0.057	0.201	0.156	0.299	0.233	0.123	0.232	**0.335**
pbmc-1C-dm	0.201	0.176	0.069	0.307	0.306	0.470	0.413	0.353	0.477	**0.529**
pbmc-2ctrl-dm	0.311	0.381	0.241	0.573	0.503	0.627	0.594	**0.675**	0.603	0.665
pbmc-2stim-dm	0.300	0.394	0.296	0.547	0.459	0.634	0.596	**0.674**	0.609	0.648
J293t-dm	0.067	0.067	0.181	**0.239**	0.189	0.103	0.158	0.175	0.192	0.230
pdx-MULTI	0.263	0.274	0.186	0.251	0.255	0.402	0.371	**0.452**	-	0.384
HMEC-orig-MULTI	0.359	0.420	0.306	0.401	0.363	0.380	0.428	0.473	**0.496**	0.383
HMEC-rep-MULTI	0.501	0.522	0.327	0.487	0.549	0.576	0.588	0.589	0.550	**0.610**
HEK-HMEC-MULTI	0.185	0.249	0.381	0.459	**0.514**	0.318	0.455	0.357	0.361	0.475
nuc-MULTI	0.217	0.260	0.107	0.356	0.367	0.355	0.383	0.294	0.422	**0.441**
mean	0.263	0.291	0.266	0.464	0.461	0.472	0.509	0.502	0.508	**0.537**

Supplementary Table S2. AUROC values of ten doublet-detection methods, including two baselines lsize and ngene, applied to 16 benchmark scRNA-seq datasets. The top-performing method on each dataset is boldfaced and underlined.

	lsize	ngene	doubletCells	Scrublet	cxds	bcds	hybrid	Solo	DoubletDetection	DoubletFinder
pbmc-ch	0.774	0.791	0.478	0.776	0.786	0.810	0.822	**0.848**	0.815	0.837
cline-ch	0.544	0.547	0.587	0.603	0.595	**0.626**	0.625	0.607	0.590	0.603
mkidney-ch	0.603	0.598	0.667	0.656	0.642	0.711	0.692	**0.754**	0.622	0.563
hm-12k	0.881	0.902	0.905	0.992	**1.000**	0.968	0.995	**1.000**	0.979	0.999
hm-6k	0.888	0.921	0.971	0.995	**1.000**	0.991	0.999	0.999	0.999	**1.000**
pbmc-1A-dm	0.781	0.787	0.532	0.726	0.807	0.828	0.834	0.808	0.787	**0.842**
pbmc-1B-dm	0.689	0.684	0.504	0.747	0.725	0.709	0.736	0.711	0.721	**0.780**
pbmc-1C-dm	0.771	0.769	0.518	0.755	0.783	0.824	0.821	0.804	0.808	**0.837**
pbmc-2ctrl-dm	0.800	0.836	0.714	0.874	0.874	0.900	0.905	**0.926**	0.906	0.917
pbmc-2stim-dm	0.797	0.846	0.732	0.865	0.856	0.898	0.898	**0.931**	0.902	0.912
J293t-dm	0.420	0.413	0.557	0.557	0.483	0.550	0.491	0.496	0.506	**0.613**
pdx-MULTI	0.640	0.644	0.593	0.643	0.657	0.741	0.725	**0.756**	-	0.701
HMEC-orig-MULTI	0.701	0.734	0.691	0.730	0.704	0.724	0.741	0.755	**0.770**	0.727
HMEC-rep-MULTI	0.644	0.663	0.512	0.646	0.693	0.698	0.710	0.717	0.689	**0.718**
HEK-HMEC-MULTI	0.767	0.784	0.732	0.759	**0.835**	0.798	0.831	0.796	0.773	0.775
nuc-MULTI	0.720	0.739	0.560	0.732	0.764	0.763	0.772	0.751	0.770	**0.794**
mean	0.714	0.729	0.641	0.753	0.763	0.784	0.787	**0.791**	0.776	0.789

Supplementary Table S3. The number of outperforming baselines and the number of top-performing for each method on 16 benchmark scRNA-seq datasets. The largest number is boldfaced and underlined.

	doubletCells	Scrublet	cxds	bcds	hybrid	Solo	DoubletDetection	DoubletFinder
# of outperforming baselines (AUPRC)	6	13	14	15	**16**	**16**	15	14
# of top-performing (AUPRC)	0	1	3	0	0	5	1	**6**
# of outperforming baselines (AUROC)	5	8	14	15	**16**	**16**	14	13
# of top-performing (AUROC)	0	0	3	1	0	6	1	**7**

Supplementary Table S4. Mean precision, recall, and true negative rates (TNRs) of ten doublet-detection methods, including the two baseline methods lsize and ngene, under three identification rates (10%, 20%, and 40%) across 16 benchmark scRNA-seq datasets. The top-performing method of each metric is boldfaced and underlined.

Identification rate	Mean	lsize	ngene	doubletCells	Scrublet	cxds	bcds	hybrid	Solo	DoubletDetection	DoubletFinder
10%	Precision	0.314	0.337	0.257	0.423	0.404	0.457	0.468	**0.476**	0.453	0.464
	Recall	0.330	0.349	0.272	0.435	0.445	0.488	**0.505**	0.498	0.481	**0.505**
	TNR	0.923	0.926	0.923	0.940	0.933	0.940	0.941	**0.942**	0.940	0.941
20%	Precision	0.254	0.275	0.208	0.289	0.290	0.324	0.326	**0.338**	0.313	0.324
	Recall	0.503	0.543	0.403	0.551	0.575	0.624	0.631	**0.636**	0.615	0.624
	TNR	0.831	0.836	0.824	0.844	0.840	0.849	0.849	0.852	0.847	**0.854**
40%	Precision	0.191	0.196	0.165	0.200	0.201	0.211	0.211	0.216	0.202	**0.219**
	Recall	0.694	0.707	0.582	0.701	0.727	0.746	0.752	**0.756**	0.738	0.734
	TNR	0.633	0.636	0.621	0.647	0.638	0.644	0.644	0.647	0.642	**0.680**

Supplementary Table S5. Precision of doublets detection on 12 benchmark scRNA-seq datasets. We executed DoubletDecon on each dataset to calculate its precision. For other methods, we calculated precision by setting up appropriate cutoffs based on the number of doublets determined by DoubletDecon. The top-performing method on each dataset is boldfaced and underlined. We excluded four datasets that DoubletDecon failed to run through.

	lsize	ngene	doubletCells	Scrublet	cxds	bcds	hybrid	Solo	DoubletDetection	DoubletFinder	DoubletDecon
pbmc-ch	0.262	0.274	0.160	0.261	0.260	0.269	0.271	**0.279**	0.263	**0.279**	0.173
cline-ch	0.214	0.214	0.245	0.250	0.241	0.259	**0.261**	0.249	0.240	0.254	0.184
mkidney-ch	0.465	0.469	0.536	0.514	0.499	0.567	0.552	**0.613**	0.472	0.446	0.373
hm-6k	0.059	0.059	0.060	**0.062**	0.061	0.061	0.061	0.061	0.061	**0.062**	0.035
pbmc-1A-dm	0.076	0.080	0.037	0.074	0.078	0.078	0.079	0.079	0.075	**0.104**	0.038
pbmc-1B-dm	0.058	0.059	0.038	0.062	0.064	0.060	0.064	0.061	0.060	**0.068**	0.031
pbmc-1C-dm	0.126	0.128	0.065	0.115	0.122	0.127	0.127	0.126	0.125	**0.159**	0.061
pbmc-2stim-dm	0.289	0.331	0.252	0.331	0.330	0.351	0.350	**0.368**	0.356	0.361	0.117
pdx-MULTI	0.197	0.197	0.171	0.202	0.201	0.247	0.237	**0.254**	--	0.229	0.131
HMEC-orig-MULTI	0.163	0.166	0.165	0.171	0.167	0.169	0.170	0.171	**0.172**	0.170	0.134
HMEC-rep-MULTI	0.333	0.337	0.319	0.336	0.345	0.345	0.348	0.348	0.338	**0.413**	0.315
HEK-HMEC-MULTI	0.110	0.112	0.102	0.104	0.118	0.115	**0.119**	0.114	0.110	0.103	0.046
mean	0.196	0.202	0.179	0.207	0.207	0.221	0.220	**0.227**	0.207	0.221	0.137

Supplementary Table S6. Recall of doublets detection on 12 benchmark scRNA-seq datasets. We executed DoubletDecon on each dataset to calculate its recall. For other methods, we calculated recall by setting up appropriate cutoffs based on the number of doublets determined by DoubletDecon. The top-performing method on each dataset is boldfaced and underlined. We excluded four datasets that DoubletDecon failed to run through.

	lsize	ngene	doubletCells	Scrublet	cxds	bcds	hybrid	Solo	DoubletDetection	DoubletFinder	DoubletDecon
pbmc-ch	0.810	0.842	0.495	0.796	0.803	0.833	0.837	**0.864**	0.815	0.861	0.536
cline-ch	0.412	0.412	0.472	0.461	0.463	0.498	**0.502**	0.480	0.461	0.453	0.355
mkidney-ch	0.495	0.499	0.570	0.545	0.532	0.604	0.588	**0.653**	0.503	0.475	0.397
hm-6k	0.965	0.971	0.982	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.573
pbmc-1A-dm	0.767	**0.808**	0.375	0.700	0.792	0.792	**0.808**	0.800	0.767	0.783	0.383
pbmc-1B-dm	0.669	0.677	0.431	0.677	**0.731**	0.685	**0.731**	0.700	0.692	**0.731**	0.354
pbmc-1C-dm	0.778	**0.788**	0.405	0.690	0.756	**0.788**	**0.788**	0.782	0.775	0.772	0.380
pbmc-2stim-dm	0.722	0.825	0.629	0.804	0.825	0.877	0.874	**0.920**	0.879	0.898	0.292
pdx-MULTI	0.519	0.519	0.451	0.527	0.532	0.651	0.626	**0.672**	--	0.569	0.347
HMEC-orig-MULTI	0.824	0.838	0.835	0.860	0.841	0.854	0.857	**0.862**	0.851	0.856	0.677
HMEC-rep-MULTI	0.856	0.866	0.822	0.861	0.887	0.887	**0.896**	0.895	0.869	0.736	0.810
HEK-HMEC-MULTI	0.701	0.718	0.652	0.663	0.755	0.734	**0.759**	0.730	0.699	0.652	0.292
mean	0.710	0.730	0.593	0.715	0.743	0.767	0.772	**0.780**	0.756	0.732	0.450

Supplementary Table S7. True negative rate (TNR) of doublets detection on 12 benchmark scRNA-seq datasets. We executed DoubletDecon on each dataset to calculate its TNR. For other methods, we calculated TNR by setting up appropriate cutoffs based on the number of doublets determined by DoubletDecon. The top-performing method on each dataset is boldfaced and underlined. We excluded four datasets that DoubletDecon failed to run through.

	lsize	ngene	doubletCells	Scrublet	cxds	bcds	hybrid	Solo	DoubletDetection	DoubletFinder	DoubletDecon
pbmc-ch	0.544	0.553	0.481	0.549	0.542	0.548	0.549	0.554	0.544	**0.556**	0.489
cline-ch	0.658	0.658	0.672	0.688	0.670	0.678	0.679	0.674	0.671	**0.699**	0.645
mkidney-ch	0.661	0.664	0.706	0.693	0.683	**0.725**	0.716	0.755	0.665	0.649	0.602
hm-6k	0.601	0.601	0.602	**0.607**	0.602	0.602	0.602	0.602	0.603	**0.607**	0.591
pbmc-1A-dm	0.646	0.648	0.632	0.669	0.645	0.645	0.646	0.646	0.644	**0.744**	0.630
pbmc-1B-dm	0.616	0.619	0.608	0.635	0.618	0.617	0.618	0.617	0.617	**0.647**	0.605
pbmc-1C-dm	0.654	0.657	0.630	0.660	0.653	0.655	0.655	0.654	0.654	**0.739**	0.628
pbmc-2stim-dm	0.764	0.779	0.752	0.784	0.778	0.785	0.784	**0.790**	0.789	0.789	0.707
pdx-MULTI	0.689	0.689	0.679	0.696	0.691	0.708	0.705	0.711	--	**0.719**	0.663
HMEC-orig-MULTI	0.341	0.343	0.343	0.349	0.344	0.346	0.346	0.347	**0.362**	0.347	0.318
HMEC-rep-MULTI	0.228	0.233	0.212	0.236	0.241	0.241	0.245	0.245	0.233	**0.529**	0.207
HEK-HMEC-MULTI	0.726	0.727	0.724	0.726	**0.729**	0.728	**0.729**	0.728	0.726	0.725	0.706
mean	0.594	0.598	0.587	0.608	0.600	0.607	0.606	0.610	0.592	**0.646**	0.566

Supplementary Table S8. The number of identified doublets by DoubletDecon compared with the true number of doublets on 12 benchmark datasets. We excluded four datasets that DoubletDecon failed to run through.

	pbmc-ch	cline-ch	mkidney-ch	hm-6k	pbmc-1A-dm	pbmc-1B-dm	pbmc-1C-dm	pbmc-2stim-dm	pdx-MULTI	HMEC-orig-MULTI	HMEC-rep-MULTI	HEK-HMEC-MULTI
# of predicted doublets	7872	2822	8417	2813	1223	1493	1961	4077	3479	18007	8448	3124
# of true doublets	2545	1465	7901	171	120	130	316	1631	1317	3568	3282	489

Supplementary Table S9. Mean running time of nine doublet-detection methods and their AUPRCs on 16 benchmark scRNA-seq datasets. The last row is the running time normalized by AUPRC. The top-performing method of each metric is boldfaced and underlined. The mean running time of DoubletDecon was calculated on 12 datasets that it ran through successfully.

	doubletCells	Scrublet	cxds	bcds	hybrid	DoubletDetection	DoubletFinder	Solo	DoubletDecon
Mean time (s)	37	64	**5**	46	47	380	243	618	903
Mean AUPRC	0.266	0.464	0.461	0.472	0.509	0.508	**0.537**	0.502	-
Time/AUPRC	137	130	**11**	97	92	749	452	1232	-

Supplementary Table S10. Mean AUPRC values of eight doublet-detection methods on benchmark scRNA-seq datasets, categorized by four experimental techniques that were used to label doublets. The mean was calculated across the datasets labeled by each technique. The top-performing method for each technique is boldfaced and underlined.

	doubletCells	Scrublet	cxds	bcds	hybrid	Solo	DoubletDetection	DoubletFinder
Cell hashing	0.342	0.483	0.479	0.532	0.536	**0.555**	0.514	0.480
Species mixture	0.499	0.949	**0.999**	0.669	0.972	0.984	0.903	0.996
Demuxlet	0.155	0.353	0.314	0.432	0.396	0.373	0.408	**0.478**
MULTI-seq	0.261	0.391	0.410	0.406	0.445	0.433	0.457	**0.459**

Supplementary Table S11. Mean Pearson correlation coefficient between every pair of doublet-detection methods in terms of their doublet scores across the 16 benchmark datasets; that is, a Pearson correlation coefficient was calculated for every pair of methods on each dataset, and the 16 coefficients were averaged into the mean coefficient for that pair.

	doubletCells	Scrublet	cxds	bcds	Solo	DoubletDetection	DoubletFinder
doubletCells	1.000						
Scrublet	0.249	1.000					
cxds	0.142	0.478	1.000				
bcds	0.109	0.455	0.642	1.000			
Solo	0.126	0.484	0.603	0.682	1.000		
DoubletDetection	0.200	0.604	0.598	0.637	0.615	1.000	
DoubletFinder	0.155	0.559	0.639	0.664	0.628	0.700	1.000

Supplementary Table S12. Mean Jaccard index between every pair of doublet-detection methods in terms of their identified doublets, whose numbers equal to the numbers of labeled doublets, across the 16 benchmark datasets; that is, a Jaccard index was calculated for every pair of methods on each dataset, and the 16 indices were averaged into the mean index for that pair.

	doubletCells	Scrublet	cxds	bcds	Solo	DoubletDetection	DoubletFinder
doubletCells	1.000						
Scrublet	0.188	1.000					
cxds	0.169	0.316	1.000				
bcds	0.152	0.290	0.397	1.000			
Solo	0.176	0.352	0.442	0.452	1.000		
DoubletDetection	0.169	0.370	0.430	0.438	0.483	1.000	
DoubletFinder	0.174	0.359	0.424	0.433	0.481	0.525	1.000

Supplementary Table S13. The default hyperparameter settings of Scrublet and DoubletFinder.

Method	Generation of artificial doublets	# of artificial doublet	# of genes to perform principal component analysis	# of principle component	k, # of nearest neighbors
Scrublet	Adding two randomly selected droplets' gene expression profiles	One-third of the # of original droplets	Top 85% highly variable genes	30	$round\ (0.5 * \sqrt{\#\ of\ droplets})$
DoubletFinder	Averaging two randomly selected droplets' gene expression profiles	Twice of the # of original droplets	Top 2000 highly variable genes	10	Selected by maximizing the mean-variance normalized bimodality coefficient of the distribution of doublet scores

CHAPTER 3

An R Package for Benchmarking Computational Doublet-Detection Methods in Single-Cell RNA Sequencing Data Analysis

3.1 Introduction

The existence of doublets is a key confounder in single-cell RNA sequencing (scRNA-seq) data analysis. There are several computational methods for detecting doublets from scRNA-seq data. We develop an R package DoubletCollection to integrate the installation and execution of those methods. DoubletCollection also provides a unified interface to perform and visualize downstream analysis after doublet detection. Here, we present a protocol of using DoubletCollection to benchmark doublet-detection methods. This protocol can automatically accommodate new doublet-detection methods in the fast-growing scRNA-seq field.

3.2 A step-by-step protocol

3.2.1 Data download

We collect 16 real scRNA-seq datasets with doublets annotated by experimental techniques. This collection covers a variety of cell types, droplet and gene numbers, doublet rates, and sequencing depths. It represents varying levels of difficulty in detecting doublets from scRNA-seq data. The data collection and preprocessing details are described in our previous work [91]. The datasets are available at Zenodo https://zenodo.org/record/4562782#.YI2lhWf0mbg in the file real_datasets.zip (Figure 1).

We save the datasets in rds format. The name of each dataset file is the same as the name defined in [91]. After being loaded into R, each dataset is a list containing two elements: the first element is a scRNA-seq count matrix with rows as genes and columns as droplets; the second element is a vector containing the singlet/doublet annotation of each droplet, which corresponds to each column in the first element.

We utilize two simulators, scDesign [51] and Splatter [67] to generate realistic scRNA-seq datasets with varying doublet rates (i.e., percentages of doublets among all droplets), sequencing depths, cell types, and between-cell-type heterogeneity levels. The synthetic datasets contain ground-truth doublets, cell types, differentially expressed (DE) genes, and cell trajectories. The simulation details are described in [91]. The datasets are available at Zenodo https://zenodo.org/record/4562782#.YI2lhWf0mbg in the file synthetic_datasets.zip (Figure 1). We save the datasets in rds format. All the synthetic datasets contain count matrices with rows as genes and columns as droplets. Below is the data structure of each dataset after being loaded into R.

sim_rate.rds: An R list with two elements. The first element contains 20 scRNA-seq count matrices that are independently generated with the doublet rates ranging from 0.02 to 0.4. Each element is named by its doublet rate and has rows as genes and columns as droplets. The second element contains 20 singlet/doublet annotation vectors, which correspond to the columns of the 20 count matrices in the first element.

sim_depth.rds: An R list with two elements. The first element contains 20 scRNA-seq count matrices that are independently generated with sequencing depth ranging from 500 to 10,000 UMI counts. Each element is named by its sequencing depth and has rows as genes and columns as droplets. The second element contains 20 singlet/doublet annotation vectors, which correspond to the columns of the 20 count matrices in the first element.

sim_type.rds: An R list with two elements. The first element contains 19 scRNA-seq count matrices that are independently generated with cell type numbers ranging from 2 to 20. Each element is named by its cell type numbers and has rows as genes and columns as droplets. The second element contains 19 singlet/doublet annotation vectors, which correspond to the columns of the 19 count matrices in the first element.

sim_hetero.rds: An R list with two elements. The first element contains 21 scRNA-seq count matrices that are independently generated with different between-cell-type heterogeneity levels as defined in [91]. Each element is named by its between-cell-type heterogeneity levels and has rows as genes and columns as droplets. The second element contains 21 singlet/doublet annotation vectors, which correspond to the columns of the 21 count matrices in the first element.

sim_clustering.rds: An R list with two elements. The first element contains three scRNA-seq count matrices with four, six, or eight cell types. Each element is named by its cell type numbers and has rows as genes and columns as droplets. The second element contains three singlet/doublet annotation vectors, which correspond to the columns of the three count matrices in the first element.

sim_DE.rds: An R list with four elements, including one synthetic scRNA-seq count matrix, its doublet indices, cell type annotations, and DE genes. This dataset contains 6% DE genes between two cell types and 40% doublets.

sim_trajectory.rds: An R list with two elements, including one synthetic scRNA-seq count matrix and its singlet/doublet annotations. This dataset contains a bifurcating cell trajectory and 20% doublets.

sim_temporally_DE.rds: An R list with three elements, including one synthetic scRNA-seq count matrix, its singlet/doublet annotations, and temporally DE genes. This dataset contains one cell trajectory with 250 temporally DE genes and 20% doublets.

Note: Please download the latest version of the datasets from the Zenodo repository. The timing for data downloading depends on the network condition.

3.2.2 Installation of DoubletCollection

DoubletCollection is an R package that integrates the installation, execution, and benchmark of eight doublet-detection methods. The source code and documentation of DoubletDetection are available at https://github.com/xnnba1984/DoubletCollection. To install DoubletDetection, execute the following R code.

```
if(!require(devtools)){
```

```
  install.packages("devtools")
}
devtools::install_github("xnnba1984/DoubletCollection")
```

Note: DoubletCollection automatically installs eight doublet-detection methods: Scrublet [40], doubletCells [48], scds [45] (including cxds, bcds, and hybrid), DoubletDetection [92], DoubletFinder [41], and scDblFinder [93]. It also installs other packages required for downstream analysis and visualization.

Optional: Solo [39] is a doublet-detection method implemented as a Linux command-line tool. DoubletCollection does not include this method. The installation and execution of Solo are available at https://github.com/calico/solo.

3.2.3 Doublet detection accuracy on real scRNA-seq datasets

This section illustrates how to apply DoubletCollection to 16 real scRNA-seq datasets, calculate the detection accuracy, and visualize the result. Every doublet-detection method in DoubletCollection outputs a doublet score for each droplet in the dataset. The larger the doublet score is, the more likely the droplet is a doublet. The following R code calculates doublet scores of user-specified methods on 16 real datasets.

```
library(DoubletCollection)

# read 16 datasets in the folder real_datasets
data.list <- ReadData(path = ".../real_datasets")
count.list <- data.list$count
```

```
# transform doublet annotations to 0/1
label.list <- lapply(data.list$label, FUN = function(label){
     ifelse(label == 'doublet', 1, 0)
})
methods <- c('doubletCells','cxds','bcds','hybrid','scDblFinder',
'Scrublet','DoubletDetection','DoubletFinder')

# calculate doublet scores
score.list.all <- FindScores.All(count.list, methods)
```

Note: All 16 rds files need to be saved under the folder real_datasets. Users can perform doublet detection on any scRNA-seq datasets by including them into count.list. Users can also choose doublet-detection methods by modifying the methods vector.

Doublet detection is essentially a binary classification problem. Therefore, the area under the precision-recall curve (AUPRC) and the area under the receiver operating characteristic curve (AUROC) are appropriate for evaluating the overall doublet-detection accuracy. The following R code calculates AUPRC and AUROC based on the doublet scores.

```
auprc.list.all <- FindAUC.All(score.list.all, label.list, 'AUPRC')
auroc.list.all <- FindAUC.All(score.list.all, label.list, 'AUROC')
```

We use boxplots to visualize the distributions of AUPRC and AUROC values of every doublet-detection method on the 16 real scRNA-seq datasets. The following R code outputs Figure 2A.

```
# transform the output of FindAUC.All to a data frame for visualization
result.auprc <- ListToDataframe(auprc.list.all, 'boxplot')
result.auroc <- ListToDataframe(auroc.list.all, 'boxplot')

# visualize AUPRC and AUROC by boxplots
Plot_Boxplot(result.auprc, 'AUPRC')
Plot_Boxplot(result.auroc, 'AUROC')
```

Note: Users can save data frames result.auprc and result.auroc to compare the AUPRC and AUROC values of doublet-detection methods.

In practice, doublets are identified based on a single threshold. To accommodate this scenario, we examine the detection accuracy of doublet-detection methods under a specific identification rate x%. For each method and each dataset, we identify the top x% droplets with the highest doublet scores as doublets. Then we calculate the corresponding precision, recall, and true negative rate (TNR). The following R code calculates precision, recall, and TNR under a 10% identification rate.

```
# call doublets based on a 10% doublet rate
doublet.list.all <- FindDoublets.All(score.list.all, rate=0.1)

# calculate precision, recall, and TNR of identified doublets
precision.list.all<- FindACC.All(doublet.list.all,label.list,'precision')
recall.list.all <- FindACC.All(doublet.list.all, label.list, 'recall')
tnr.list.all <- FindACC.All(doublet.list.all, label.list, 'TNR')
```

Optional: Users can calculate the precision, recall, and TNR under varying doublet rates to conduct a more comprehensive comparison of doublet-detection methods.

Again, we use boxplots to visualize the distributions of precision, recall, and TNR values of each method under specific identification rates. The following R code outputs Figure 2B.

```
# transform the output of FindAcc.All to a data frame for visualization
result.precision <- ListToDataframe(precision.list.all, 'boxplot')
result.recall <- ListToDataframe(recall.list.all, 'boxplot')
result.tnr <- ListToDataframe(tnr.list.all, 'boxplot')

# visualize precision, recall, and TNR by boxplots
Plot_Boxplot(result.precision, 'Precision')
Plot_Boxplot(result.recall, 'Recall')
Plot_Boxplot(result.tnr, 'TNR')
```

Note: Users can save data frames result.precision, result.recall, and result.tnr to compare the precision, recall, and TNR values of doublet-detection methods.

3.2.4 Hyperparameter tuning for doublet detection methods (Optional)

The previous R code sets the hyperparameters of doublet-detection methods to their recommended or default values. This section explains how to use DoubletCollection to search for the hyperparameters that may potentially improve the doublet-detection methods' performance.

We set up a series of hyperparameter values and use DoubletCollection to conduct a grid search. DoubletCollection returns a combination of hyperparameters that optimizes a user-specified accuracy measure on a dataset. The following R code searches for optimal hyperparameters in terms of AUPRC for the methods Scrublet, DoubletFinder, and scDblFinder on the dataset pbmc-1A-dm.

```
data.list <- ReadData(path = ".../real_datasets")
count.list <- data.list$count
label.list <- lapply(data.list$label, FUN = function(label){
  ifelse(label == 'doublet', 1, 0)
})

# read dataset
count <- count.list$`pbmc-1A-dm`
label <- label.list$`pbmc-1A-dm`

# search for optimal hyperparameters of Scrublet
result.parameter.Scrublet <- FindParameters(count, label, method =
'Scrublet', type = 'AUPRC', n_neighbors = c(27, 28, 29, 30, 31),
                            n_prin_comps = c(20, 25, 30, 35, 40),
                      min_gene_variability_pctl = c(60, 65, 70, 85, 90))

# search for optimal hyperparameters of DoubletFinder
result.parameter.DoubletFinder <- FindParameters(count, label,
method = 'DoubletFinder', type = 'AUPRC',
                           nfeatures = c(1000, 1500, 2000, 2500, 3000),
                           PCs = c(10, 15, 20, 25, 30))
```

```
# search for optimal hyperparameters of scDblFinder
result.parameter.scDblFinder <- FindParameters(count, label,
method = 'scDblFinder', type = 'AUPRC',
                            nf=c(500, 1000, 1500, 2000, 2500),
                            includePCs=c(3, 4, 5, 6, 7),
                            max_depth=c(3, 4, 5, 6, 7))
```

Note: Users can search for optimal hyperparameters of other doublet-detection methods on any datasets. The searchable hyperparameters of a doublet-detection method can be shown by executing ?FindParameters.

The optimal hyperparameters found from a representative dataset provide guidance for applying a doublet-detection method to similar datasets. The following R code sets the hyperparameters of three doublet-detection methods, which are to be applied to the dataset pbmc-1B-dm, to their optimal values found from the dataset pbmc-1A-dm. These two datasets share the same cell types and experimental protocol.

```
score.list <- FindScores(count = count.list$`pbmc-1B-dm`,
         methods = c('Scrublet','DoubletFinder','scDblFinder'),
         n_neighbors=31, min_gene_variability_pctl=60, n_prin_comps=40,
         nfeatures=1000, PCs=10, nf=1000, includePCs=6, max_depth=5)
```

Note: Users can also adjust hyperparameters based on their prior knowledge. The R code in the following sections uses the recommended or default hyperparameter values of doublet-detection methods.

3.2.5 Doublet detection accuracy under various experimental settings and biological conditions

This section illustrates how to apply DoubletCollection to synthetic scRNA-seq datasets under a wide range of experimental settings and biological conditions, calculate the detection accuracy of different doublet-detection methods, and visualize the result.

As in the previous section, we first calculate doublet scores on synthetic datasets. The following R code calculates doublet scores on the dataset sim_rate with different doublet rates.

```
data.list <- readRDS(".../synthetic_datasets/sim_rate.rds")
count.list <- data.list$count
label.list <- lapply(data.list$label, FUN = function(label){
  ifelse(label == 'doublet', 1, 0)
})
score.list.all <- FindScores.All(count.list, methods)
```

Note: Users can read datasets sim_depth, sim_type, or sim_hetero to calculate doublet scores under various sequencing depth, number of cell types, or degree of between-cell-type heterogeneity. The code in the following sections can be applied to those datasets without modification.

Similar to real datasets, we use AUPRC and AUROC to measure the overall detection accuracy on synthetic datasets. The following R code calculates AUPRC and AUROC based on the doublet scores obtained from the previous step.

```
auprc.list.all <- FindAUC.All(score.list.all, label.list, 'AUPRC')
auroc.list.all <- FindAUC.All(score.list.all, label.list, 'AUROC')
```

We use line plots to show how the performance of each doublet-detection method changes when we vary the experimental settings and biological conditions. The following R code draws line plots for AUPRC and AUROC under varying doublet rates. Figure 3A shows the AUPRC and AUROC values of different doublet-detection methods under different doublet rates, sequencing depths, numbers of cell types, and heterogeneity between cell types.

```
# transform the output of FindAuc.All to a data frames for visualization
result.auprc <- ListToDataframe(auprc.list.all, 'lineplot')
result.auroc <- ListToDataframe(auroc.list.all, 'lineplot')

# visualize AUPRC and AUROC by line plots
Plot_Lineplot(result.auprc, 'Doublet Rate', 'AUPRC')
Plot_Lineplot(result.auroc, 'Doublet Rate', 'AUROC')
```

3.2.6 Effects of doublet detection on DE gene analysis

This section illustrates how to use DoubletCollection to conduct differentially expressed (DE) gene analysis. We compare the results of DE gene analysis on the contaminated dataset (with 40% doublets), the clean dataset (without doublets), and the dataset after each doublet-detection method is applied.

We first read in the dataset sim_DE that includes the ground-truth DE genes and 40% doublets. Then we apply doublet-detection methods to obtain doublet scores. Finally, we remove the top 40% droplets that receive the highest doublet scores from each method.

```
data.de <- readRDS('.../synthetic_datasets/sim_DE.rds')
score.list <- FindScores(data.de$count, methods)
doublet.list <- FindDoublets(score.list, rate=0.4)

# add the clean data matrix to the data list
doublet.list[['Clean Data']] <- data.de$label.doublet

# remove identified doublets
data.removal.list <- RemoveDoublets.Method(data.de$count,
data.de$label.cluster, doublet.list)

# add original contaminated data to the data list
data.removal.list[['Contaminated Data']] <- list(count=data.de$count,
label=data.de$label.cluster)
```

We use the Wilcoxon rank-sum test [57], MAST [56], and likelihood-ratio test [94] (bimod) to identify DE genes between two cell types. The accuracy of DE gene identification is measured by precision, recall, and TNR.

```
# create a data frame to save result for visualization
table.DE.all <- data.frame()

# use three DE methods
```

```
for(DE.method in c('MAST', 'wilcox', 'bimod')){

# identify DE genes
DE.list <- FindDE(data.removal.list, DE.method)

# calculate precision, recall, and TNR of identified DE genes
DE.acc.list <- FindDEACC(DE.list, data.de$gene.de, rownames(data.de$count))

# transform to a data frame for visualization
   table.DE <- ListToDataframe(DE.acc.list, 'barplot')
   table.DE[['DE_method']] <- DE.method
   table.DE.all <- rbind(table.DE.all, table.DE)
}
```

Note: Users can choose from seven DE methods by specifying the second parameter of the function FindDE, including 'wilcox', 'bimod', 't', 'poisson', 'negbinom', 'LR', and 'MAST'. A detailed demonstration of those methods is available at https://satijalab.org/seurat/articles/de_vignette.html.

We use barplots to compare the results of DE gene analysis on the contaminated dataset (negative control), the clean dataset (positive control), and post-doublet-detection datasets. The following R code outputs barplots that compare the precision, recall, and TNR in Figure 3B. Each barplot stacks the results of three DE methods: Wilcoxon rank-sum test, MAST, and likelihood-ratio test (bimod).

```
Plot_Barplot(table.DE.all[table.DE.all$measurement=='precision',],
'Precision')
```

```
Plot_Barplot(table.DE.all[table.DE.all$measurement=='recall',], 'Recall')
Plot_Barplot(table.DE.all[table.DE.all$measurement=='tnr',], 'TNR')
```

3.2.7 Effects of doublet detection on cell clustering

This section illustrates how to use DoubletCollection to evaluate the effects of doublet-detection methods on cell clustering. First, we examine the efficacy of doublet-detection methods for removing spurious cell clusters formed by doublets. Second, we compare the proportion of singlets in the correctly identified cell clusters after each doublet-detection method is applied.

We first read the dataset sim_clustering that includes three datasets with four, six, and eight cell types and 20% doublets. Then we apply doublet-detection methods to obtain doublet scores and remove doublets based on various doublet rates.

```
data.list <- readRDS(".../synthetic_datasets/sim_clustering.rds")
count.list <- data.list$count
label.list <- lapply(data.list$label, FUN = function(label){
  ifelse(label == 'doublet', 1, 0)
})
score.list.all <- FindScores.All(count.list, methods)

# call doublets based on doublet rates from 0.01 to 0.25
doublet.list.all.rate <- FindDoublets.All.Rate(score.list.all,rates =
seq(0.01, 0.25, 0.01))

# remove identified doublets under different doublet rates
```

```
data.removal.all.rate <- RemoveDoublets.All.Rate(count.list, label.list,
doublet.list.all.rate)
```

We apply Louvain clustering [63] to the post-doublet-removal datasets to identify cell clusters.

```
result.cluster.all.rate <- Clustering.All.Rate(data.removal.all.rate)
```

We use heatmaps to compare the efficacy of doublet-detection methods for removing spurious cell clusters. The following R code outputs heatmaps of clustering results on datasets with four, six, and eight cell clusters under various doublet rates (Figure 4A).

```
# transform the output of Clustering.All.Rate to a data frame for
# visualization
table.cluster <- ListToDataframe(result.cluster.all.rate, type='heatmap')

# draw heatmaps of clustering results
Plot_Heatmap(table.cluster, cluster = 4)
Plot_Heatmap(table.cluster, cluster = 6)
Plot_Heatmap(table.cluster, cluster = 8)
```

Homotypic doublets tend to cluster together with singlets and thus do not form spurious clusters. To evaluate the efficacy of doublet-detection methods for eliminating homotypic doublets, we calculate the proportion of singlets in each identified cell cluster when the number of cell clusters matches the number of cell types.

```
table.cluster.quality <- Clustering.Quality(table.cluster,
                      result.cluster.all.rate, data.removal.all.rate)
```

We use boxplots to visualize the singlet proportions within clusters after applying doublet-detection methods, if the remaining droplets lead to the correct number of cell clusters (Figure 4B).

```
Plot_Boxplot(table.cluster.quality[table.cluster.quality$correct=='4',],
'Singlet Rates (Four Clusters)')
Plot_Boxplot(table.cluster.quality[table.cluster.quality$correct=='6',],
'Singlet Rates (Six Clusters)')
Plot_Boxplot(table.cluster.quality[table.cluster.quality$correct=='8',],
'Singlet Rates (Eight Clusters)')
```

3.2.8 Effects of doublet detection on cell trajectory inference

This section illustrates how to use DoubletCollection to evaluate the effects of doublet-detection methods on cell trajectory inference. First, we examine the efficacy of doublet-detection methods for removing spurious cell branches formed by doublets. Second, we compare the accuracy of temporally DE gene identification after doublet-detection methods are applied.

We use Slingshot [68] to infer the cell trajectories on the dataset sim_trajectory. It contains two cell branches mixed with 20% doublets (contaminated dataset). The following R code shows a two-dimensional visualization of the inference result. It contains

three cell trajectories instead of two, and the intermediate trajectory is formed by doublets (Figure 5A).

```
data.trajectory <- readRDS('.../synthetic_datasets/sim_trajectory.rds')
count <- data.trajectory$count
label <- data.trajectory$label

# cell trajectory inference by Slingshot and visualization
FindTrajectory(count, label, title='Contaminated Data')
```

We use Slingshot to infer the cell trajectories on the dataset sim_trajectory after removing all 20% doublets (clean dataset). The following R code shows a two-dimensional visualization of the inference result with two correct cell trajectories (Figure 5A).

```
# remove all doublets
count.clean <- count[,which(label==0)]
label.clean <- label[which(label==0)]

# cell trajectory inference by Slingshot and visualization
FindTrajectory(count.clean, label.clean, title='Clean Data')
```

We first perform doublet detection on the dataset sim_trajectory to obtain doublet scores. Then for each method, we remove the top 20% droplets that receive the highest doublet scores. Finally, we infer and visualize cell trajectories on each post-doublet-

removal dataset to examine if the corresponding doublet-detection method removes the spurious cell branches formed by doublets (Figure 5A).

```
score.list <- FindScores(count, methods)
doublet.list <- FindDoublets(score.list, rate = .2)
data.removal.list <- RemoveDoublets.Method(count, label,doublet.list)

# infer trajectory on each post-doublet-removal dataset
for(method in methods){
   FindTrajectory(data.removal.list[[method]]$count,
                  data.removal.list[[method]]$label, title = method)
}
```

We first use Slingshot to infer the cell pseudotime on the dataset sim_temporally_DE (contaminated dataset). It contains a single cell lineage with 250 temporally DE genes out of 750 genes, mixed with 20% doublets. Second, we use a general additive model (GAM) [83] to regress each gene's expression levels on the inferred pseudotime. Finally, we calculate the precision, recall, and TNR of the inferred temporally DE genes identified using the Bonferroni-corrected p-value threshold of 0.05. We repeat the same analysis on the clean dataset (without doublets) and each post-doublet-removal dataset.

```
data.trajectory <-readRDS('.../synthetic_datasets/sim_temporally_DE.rds')
count <- data.trajectory$count
label <- data.trajectory$label

# ground-truth temporally DE genes
```

```
gene.de <- data.trajectory$gene.de

# calculate precision, recall, and TNR of temporally DE genes for
# contaminated data
de.temp.list <- FindTempDE(count, gene.de)

# calculate doublet scores and remove doublets
score.list <- FindScores(count, methods)
count.clean <- count[,which(label==0)]
label.clean <- label[which(label==0)]
doublet.list <- FindDoublets(score.list, rate=0.2)
data.removal.list <- RemoveDoublets.Method(count, label, doublet.list)

# add clean data
data.removal.list[['Clean Data']] <- list(count.clean, label.clean)

# calculate precision, recall, and TNR of temporal DE genes for
# post-doublet-removal data
de.temp.result.all <- FindTempDE.All(data.removal.list, gene.de)

# add the result of contaminated data
de.temp.result.all[['Contaminated Data']] <- de.temp.list
```

We use barplots to compare the results of temporally DE genes identification on the contaminated dataset, the clean dataset, and the post-doublet-removal datasets (Figure 5B). The barplot stacks the results of precision, recall, and TNR for different doublet-detection methods.

```
# transform to data frame for visualization
table.DE.temp <- ListToDataframe(de.temp.result.all, type='barplot')

# draw barplot
Plot_Barplot_temp(table.DE.temp, title='Temporally DE Genes')
```

3.2.9 Performance of doublet-detection methods under distributed computing

This section illustrates how to use DoubletCollection to evaluate the accuracy of doublet-detection methods under distributed computing. This benchmark simulates the scenario when the large scRNA-seq dataset is beyond the capacity of a single computer so that the dataset must be divided into subsets to be analyzed in parallel.

First, we randomly split the dataset pbmc-ch into two up to ten equal-sized batches. Second, for each batch number, we execute every doublet-detection method on each batch separately and concatenate the resulting doublet scores across batches. Finally, we calculate the distributed AUPRC based on the concatenated doublet scores.

```
# read dataset pbmc-ch
data.list <- ReadData(path = ".../real_datasets")
count <- data.list$count$`pbmc-ch`
label <- data.list$label$`pbmc-ch`
label <- ifelse(label == 'doublet', 1, 0)
```

```
# calculate distributed AUPRC for different methods
auc.list.batch <- FindDistributedAUC.All(count,label,methods,
batches=2:10, type='AUPRC')
```

We use line plots to show how the detection accuracy of each method changes as the number of batches increases. The following R code places the batch numbers on the x-axis and connects AUPRC values to show the trend of each method (Figure 5C).

```
# transform the output of FindDistributedAUC.All to a data frame for
# visualization
table.batch <- ListToDataframe(auc.list.batch, type='distributed')

# draw line plots
Plot_Lineplot_Distributed(table.batch,data='pbmc-ch',measurement='AUPRC')
```

Optional: Users can apply the same pipeline to evaluate the detection accuracy of doublet-detection methods under distributed computing on any other real datasets.

3.2.10 Computational aspects of doublet-detection methods (optional)

The benchmark of computational aspects of doublet-detection methods includes but is not limited to efficiency, scalability, stability, and software implementation. First, we can summarize the running time of doublet-detection methods on the 16 real scRNA-seq datasets. The result can be visualized by boxplots similar to Figure 2A to compare the computational efficiency of doublet-detection methods. Second, we can examine how fast each method's running time increases as the number of droplets grows. The result can

be visualized by line plots similar to Figure 3A to examine the scalability of doublet-detection methods. Third, we can evaluate how much each method's AUPRC and AUROC values vary across subsets of droplets and genes. The result can be visualized by violin plots to compare the statistical stability of doublet-detection methods. Finally, we can qualitatively evaluate the software implementation of doublet-detection methods from the aspects of user-friendliness, software quality, and active maintenance. The complete visualization details are available in [91].

3.3 Expected Outcomes

The major outcomes of this protocol are the measures of doublet-detection accuracy and the result of downstream analysis, including AUPRC, AUROC, precision, recall, TNR, number of cell clusters, cell trajectories, DE genes, and their visualization. These outcomes are in the intermediate outputs of the R code shown in previous sections. The visualizations are shown in Figures 2 to 5. More visualizations, tables, and interpretations are available in our previous work [91].

Another important result in this protocol paper is the benchmark of a new method scDblFinder, which was not included in the previous benchmark study [91]. On the 16 real RNA-seq datasets, scDblFinder achieves the highest mean AUPRC and AUROC values, and it is also the top method in terms of precision, recall, and TNR under the 10% identification rate. On the synthetic RNA-seq datasets, scDblFinder exhibits similar performance trends to those of other doublet-detection methods under various experimental settings and biological conditions, and it is also a near-top method in terms of AUPRC. In particular, scDblFinder is able to consistently improve downstream

analyses, including DE gene, cell clustering, and cell trajectory inference. Similar to other doublet-detection methods, scDblFinder has decreased detection accuracy as the number of batches increases under distributed computing. scDblFinder is also one of the fastest doublet-detection methods (the comparison of running time is not shown). Overall, scDblFinder has excellent detection accuracy and high computational efficiency.

3.4 Limitations

The first limitation of this protocol is that the current benchmark results are based on the default hyperparameters of doublet-detection methods [24,91]. Therefore, the benchmark results in this protocol may have underestimated the performance of some doublet-detection methods. With the functionality of hyperparameter tuning provided in the R package DoubletCollection, users can conduct an independent study to explore the optimal hyperparameters of doublet-detection methods.

The second limitation of this protocol is that the doublet annotations in the 16 real scRNA-seq datasets are not completely accurate due to experimental limitations. For example, datasets hm12k and hm6k only labeled the heterotypic doublets formed by a human cell and a mouse cell [52]; datasets generated by demuxlet only labeled the doublets formed by cells of two individuals [43]; many homotypic doublets were unlabeled in real datasets [91]. The incompleteness of doublet annotations would have inflated the false negative rates and reduced the precision of computational doublet-detection methods. The synthetic datasets used in this protocol contain ground-truth doublets and thus can partly alleviate this issue.

The third limitation of this protocol is that it mainly focuses on doublet-detection methods that can generate a doublet score for every droplet in the dataset. Among currently available doublet-detection methods, DoubletDecon directly outputs identified doublets without providing doublet scores. To fairly compare DoubletDecon with other methods, we suggest users to first execute it on every dataset and record its number of identified doublets; then users can threshold the doublet scores of the other methods so that every method identifies the same number of doublets as DoubletDecon does; finally, users can calculate the precision, recall, and TNR based on the doublets identified by each method from every dataset. A detailed comparison between DoubletDecon and other methods has been discussed by [91]. Guidance for executing DoubletDecon is available at [46,95].

3.5 Troubleshooting

Problem 1: Method Scrublet or DoubletDetection fails to be installed even with a Python environment installed in the system.

Potential Solution: This problem typically happens in the Windows system with error information “Microsoft Visual C++ 14.0 is required”. To solve this problem, users can download and install the latest version of Visual Studio Build Tools at https://visualstudio.microsoft.com/downloads/ under the menu “Tools for Visual Studio 2019 -> Build Tools for Visual Studio 2019”.

Problem 2: The installation of DoubletCollection fails to install some dependent packages.

Potential Solution: This problem is caused by the version conflict when updating certain packages already installed in R. We recommend users skip the updating step suggested by R. Ignoring the update of dependent packages will not affect the functionality of DoubletCollection.

Problem 3: Some methods fail to generate doublet scores on 16 real scRNA-seq datasets.

Potential Solution: This problem is caused by memory shortage when executing certain methods on large-scale datasets. For example, we observed such issues for DoubletFinder on a laptop with 16GB memory. However, using the same code and data, the issue disappears on a server with 256GB memory. To successfully replicate the result in this protocol, we suggest users execute DoubletCollection on a computer with 64GB or more memory. If users perform doublet detection on smaller datasets, then the memory size requirement is less.

Problem 4: The ReadData function cannot read scRNA-seq datasets into the R environment.

Potential Solution: The ReadData function is designed to read all rds files under the user-specified directory. Therefore, users need to save all rds files in the directory indicated by the path parameter of ReadData. Users can also use the generic R function readRDS to read the single rds file.

Problem 5: The p-values of the hypergeometric test (which are also doublet scores) output by DoubletDetection are negative.

Potential Solution: This problem occasionally happens and is likely due to the numerical overflow of DoubletDetection. We suggest users add the abs() function outside

the output doublet scores to fix this issue. Our experiments find that DoubletDetection performs well under this correction.

3.6 Figures

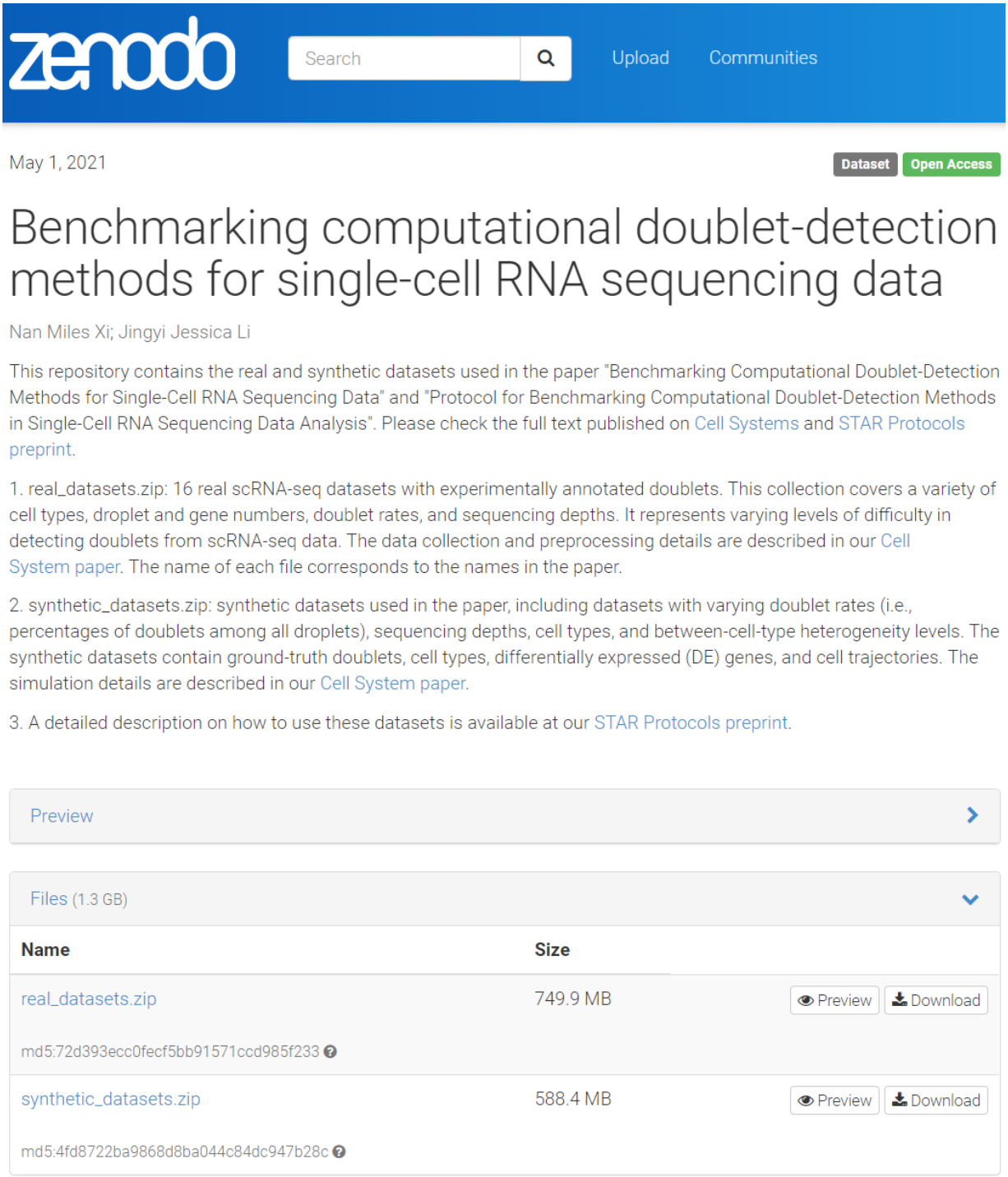


Figure 1. The Zenodo repository for downloading real and synthetic scRNA-seq datasets used in this protocol.

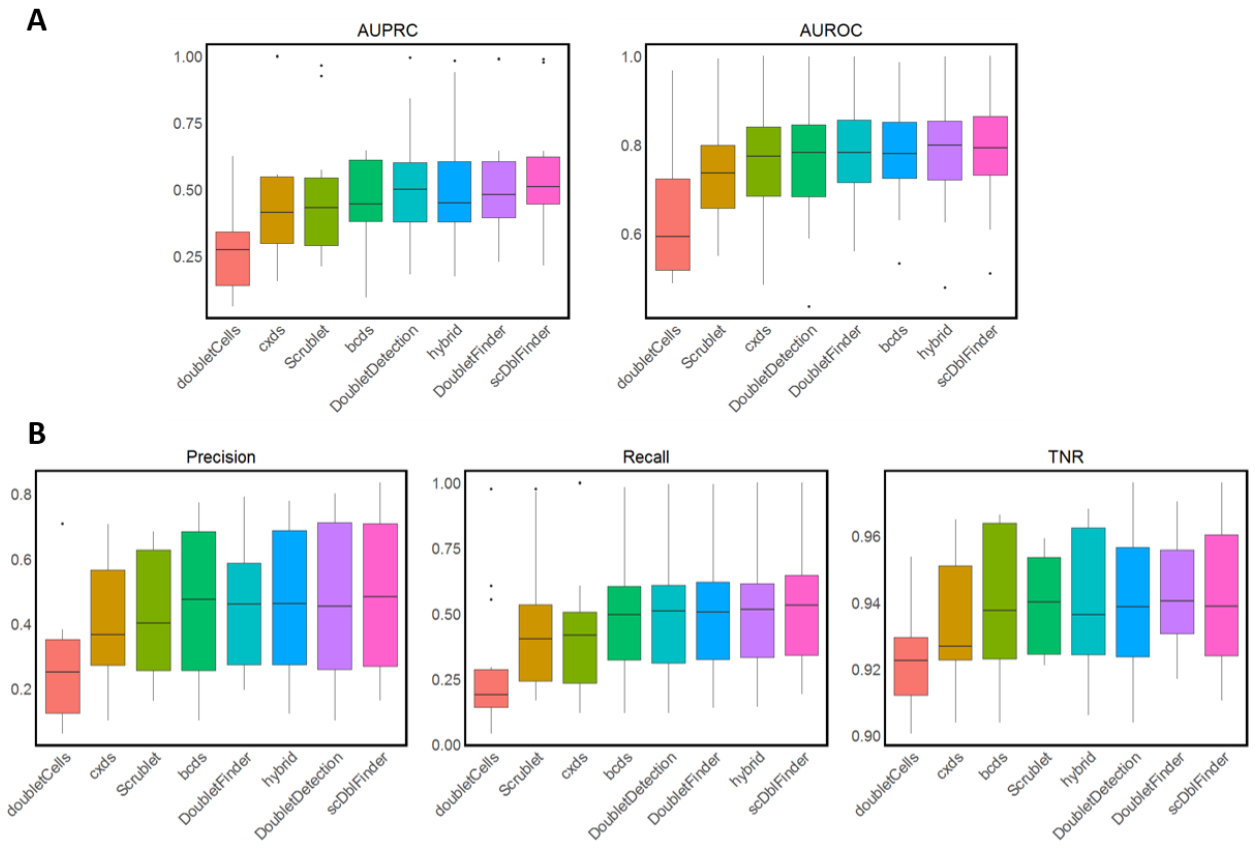


Figure 2. Evaluation of Doublet-Detection Methods Using 16 Real scRNA-Seq Datasets. (A) AUPRC and AUROC values of each method applied to 16 datasets. (B) Precision, recall, and TNR values of each method under the 10% identification rate. Methods are ordered by their average performance measurement across 16 datasets (low to high).

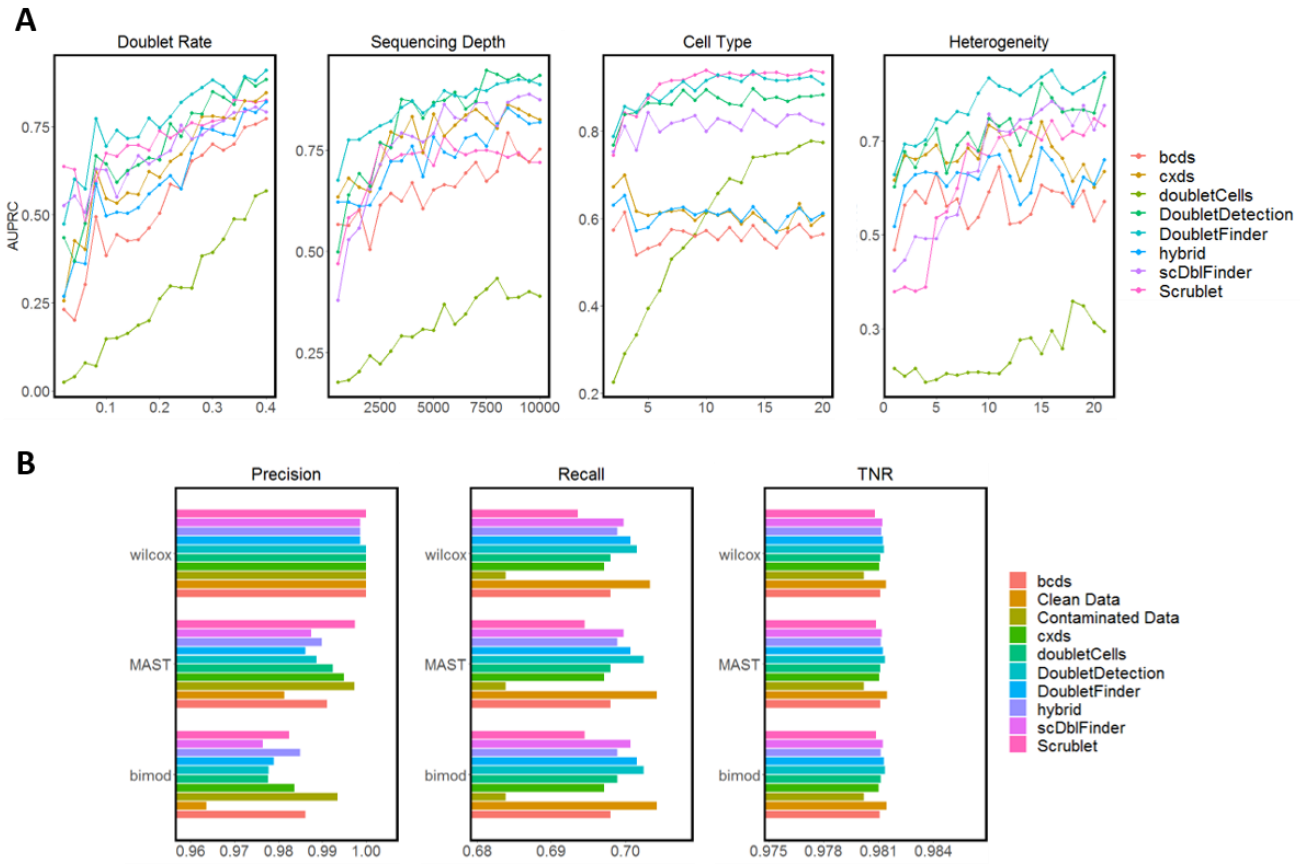


Figure 3. Evaluation of Doublet-Detection Methods Using Four Simulation Studies, and the Effects of Doublet Detection on DE Gene Analysis. (A) AUPRC of each method in four simulation settings: varying doublet rates (from 2% to 40% with a step size of 2%), varying sequencing depths (from 500 to 10,000 UMI counts per cell, with a step size of 500 counts), varying numbers of cell types (from 2 to 20 with a step size of 1), and 20 heterogeneity levels, which specify the extent to which genes are differentiated between two cell types. (B) Precision, recall, and TNR by each of three DE methods: Wilcoxon rank-sum test (wilcox), MAST, and likelihood-ratio test (bimod) after each doublet-detection method is applied to a simulated dataset; for negative and positive controls, we included the DE accuracies on the contaminated data with 40% doublets and the clean data without doublets.

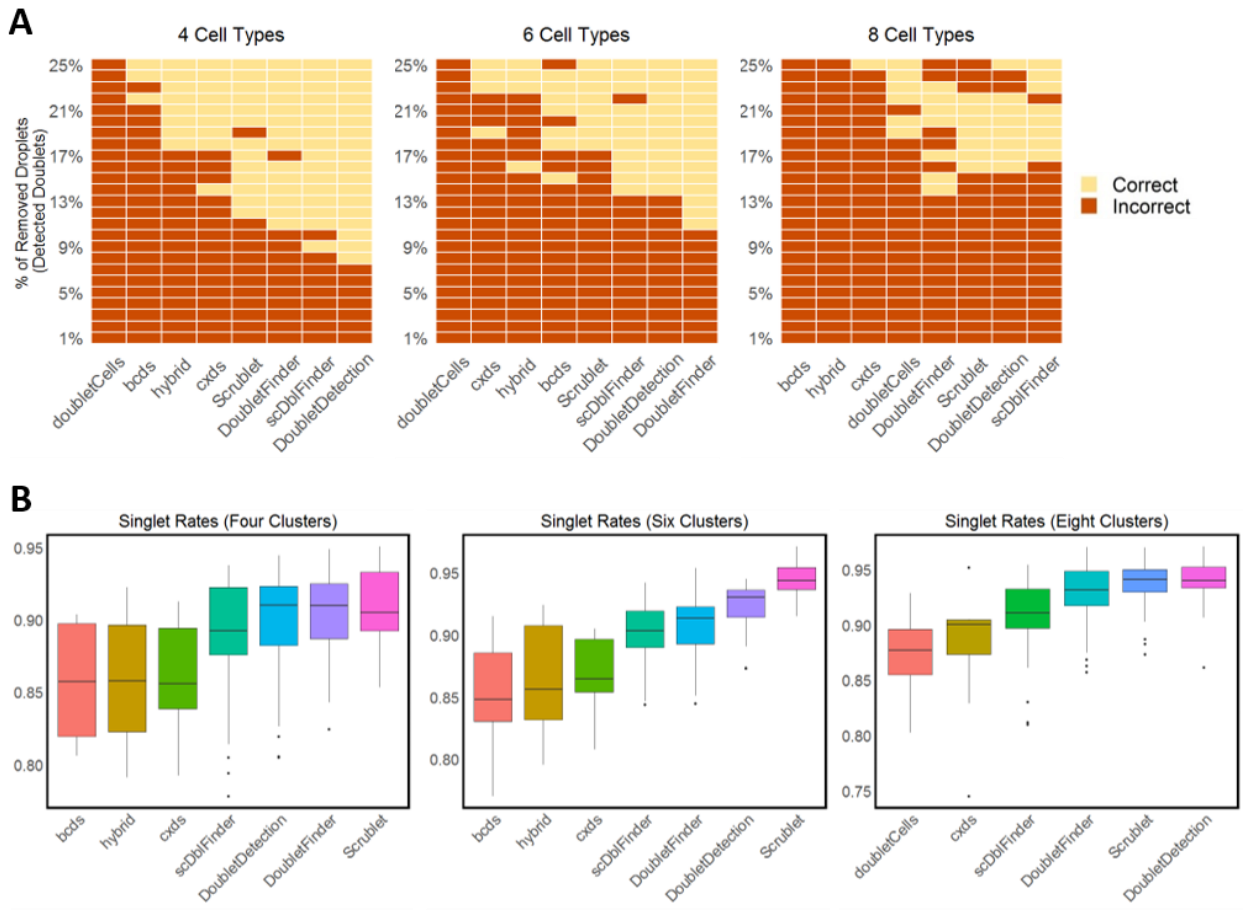


Figure 4. The Effects of Doublet Detection on Cell Clustering. (A) Cell clustering results by the Louvain algorithm after each doublet-detection method is applied to remove a varying percentage of droplets as the identified doublets (y-axis, from 1% to 25% with a step size of 1%); the true numbers of cell clusters are four, six, and eight under three simulation settings, each containing 20% true doublets; the yellow color indicates that the correct number of clusters was identified, while the red color indicates otherwise. (B) Under the same three simulation settings as in (A), the distributions of the singlet proportions are shown after doublet removal by each method, if the remaining droplets lead to the correct number of cell clusters in (A); some methods are not shown because they do not lead to the correct number of cell clusters in (A). Methods are ordered by their average performance measurement across 16 datasets (low to high).

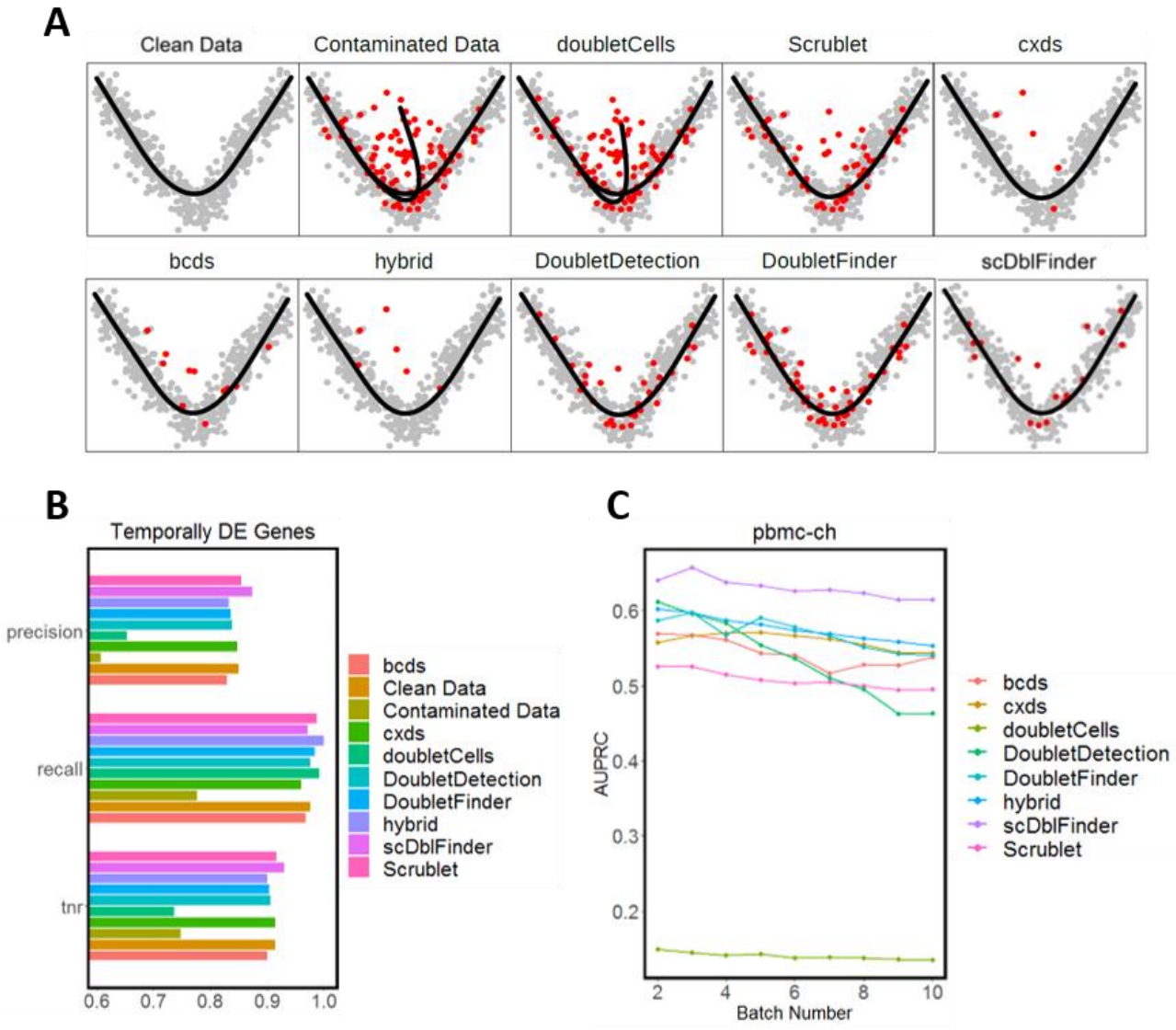


Figure 5. Effects of Doublet Detection on Cell Trajectory Inference and the detection accuracy under distributed computing. (A) Cell trajectories constructed by Slingshot. (B) Precision, recall, and TNR of temporally DE genes inferred by the GAM. Both (A) and (B) are performed on contaminated, clean, and post-doublet-detection datasets. (C) AUPRC of each doublet detection method on the real dataset pbmc-ch under distributed computing.

CHAPTER 4

Benchmarking the Design of Deep Autoencoders for Denoising Single-Cell RNA Sequencing Data

4.1 Introduction

Single-cell RNA-sequencing (scRNA-seq) enables the measurement of genome-wide gene expression at single-cell levels [8,9,96]. scRNA-seq can generate datasets with tens of thousands of genes and up to millions of cells [14], which allows for the investigation of cell-to-cell heterogeneity [97], identification of distinguished cell type [98], and quantification of cell state transition [99]. One characteristic of scRNA-seq data is the high proportion of zeros or high sparsity. Depending on the sequencing platform and sequencing depth, the zero proportion of one scRNA-seq data matrix ranges from 50% to more than 90% [24]. There are two types of zeros in scRNA-seq data — biological zeros and non-biological zeros [26]. Biological zeros indicate the actual absence of gene expression in the cell, while non-biological zeros originate from the technical limitation or noise in the scRNA-seq experiment [100]. Without external reference or prior biological knowledge, it is usually

difficult to distinguish between these two types of zeros in scRNA-seq data [24]. In the following text, we will interchangeably use the terminology non-biological zeros, technical zeros, and missing values.

The high sparsity of scRNA-seq data, especially the non-biological zeros, poses a great challenge in downstream analysis of scRNA-seq data [24]. Many computational methods have been developed to impute and denoise the non-biological zeros (sometimes also biological zeros), and they can be divided into three broad categories. First, model-based imputation methods infer probabilistic models to capture the distribution of gene expression in scRNA-seq data. They aim to differentiate between biological zeros and non-biological zeros and only impute the former [27,28]. Second, data-smoothing methods modify the gene expression in each cell based on the similar cells in the datasets. The similarity is defined by the neighborhood of each cell in a low dimensional space [29,30]. Data-smoothing methods impute all zeros and also change the non-zero values in scRNA-seq data. Third, data-reconstruction methods utilize machine learning techniques to learn a latent space of the original sparse data. The latent space is further used to reconstruct the imputed dense data [31,32]. Data-reconstruction methods also impute all zeros but keep non-zero values unchanged in scRNA-seq data.

Recently, autoencoder-based imputation methods have gained much attention due to their superior imputation accuracy, a large improvement on downstream analysis, high degree of flexibility, and capacity of extending to large-scale datasets [22,33]. These methods belong to the data-reconstruction category and use neural networks to learn a latent space and reconstruct the imputed scRNA-seq data. Specifically, an autoencoder contains one encoder neural network and one decoder neural network — the encoder

neural network first compresses the high-dimensional scRNA-seq data into a low-dimensional latent space; then a decoder neural network recovers the data to its original dimension from the latent space. The recovered data is used to impute the sparse scRNA-seq data. One challenge when designing an autoencoder-based imputation method is how to select its hyperparameters, including the neural network structure, activation function, and regularization [24]. The existing methods either copy the practice in other fields, especially the computer vision study, or set up the autoencoder on an ad hoc basis [31,32,103-106]. Currently, there is no formal discussion on the choice of hyperparameters in the autoencoder for imputing scRNA-seq data.

Here, we conduct the first empirical study to systematically explore the best modeling strategies for autoencoder-based scRNA-seq imputation methods. In particular, we first design three masking schemes to introduce ground-truth non-biological zeros on 12 real scRNA-seq datasets. Second, we train autoencoders with a large variety of depth and width, seven activation functions, and two types of regularization on those datasets, and further impute them to calculate the imputation accuracy based on the masked values. We compare the imputed normalized root mean square error (NRMSE) and imputed Pearson correlation coefficient of different autoencoders to examine the impact of their design on the overall imputation accuracy. Third, we train autoencoders with the aforementioned hyperparameter settings on 20 real scRNA-seq datasets with ground-truth cell type information. Then we conduct cell clustering on pre-imputed and imputed datasets after applying different autoencoders. We compare the adjusted Rand index (ARI) and adjusted mutual information (AMI) to examine the impact of autoencoder design on downstream cell clustering. Fourth, we simulate 20 synthetic datasets with ground-

truth differentially expression (DE) genes based on 20 real scRNA-seq datasets. Again, we train autoencoders with the aforementioned hyperparameter settings on those synthetic datasets and identify DE genes on pre-imputed and imputed datasets after applying different autoencoders. We compare the precision, recall, and true negative rate (TNR) of identified DE genes to examine the impact of autoencoder design on downstream DE gene analysis.

Our analysis shows results that are largely ignored in the previous method development. First, deeper autoencoders provide better overall imputation accuracy, cell clustering, and DE gene analysis. The benefits from depth generally saturate when the autoencoder passes 10 hidden layers. Second, narrower autoencoders improve the overall imputation accuracy but are similar to wider autoencoders in terms of cell clustering and DE gene analysis. Third, the sigmoid and tanh activation functions consistently outperform others in all evaluations. Fourth, the weight decay and dropout regularization are critical to the performance of autoencoder-base imputation methods. In particular, weight decay is more capable of improving cell clustering and DE gene analysis while dropout shows superiority in improving overall imputation accuracy. The optimal hyperparameters of these two regularizations largely depend on datasets. Our findings contradict the common practice in previous methods, where shallower and wider autoencoder with ReLU activation functions are widely used. Those findings highlight the unique characteristics of scRNA-seq data in imputation tasks and call cautions on borrowing model design directly from other fields. Our empirical study provides insights for the future development of autoencoder-based imputation methods for scRNA-seq data.

4.2 Results

4.2.1 Autoencoder for imputing scRNA-seq data

An autoencoder is a multi-layer neural network that aims to reconstruct the input data through its hidden layers [101]. When being applied to imputing the scRNA-seq data, the autoencoder is able to learn a low-dimensional representation of the input data and use it to recover the missing values (Figure 1a). Let X be the sparse scRNA-seq input matrix after appropriate preprocessing and normalization (Methods). Let Y be the dense scRNA-seq output matrix. Both X and Y have n rows (cells) and m genes (columns). Suppose H_k is the kth hidden layer of the autoencoder, where $k = 1, 2, \ldots, h$ and h is the number of hidden layers. Then the first hidden layer H_1 is calculated as

$$H_1 = f\ (XW_1 + b_1),$$

where W_1 is an m by l_1 weight matrix and b_1 is an $l_1$dimensional bias vector. f is an element-wise nonlinear activation function. Similarly, the $k+1$th hidden layer H_{k+1} is calculated as

$$H_{k+1} = f\ (H_k W_{k+1} + b_{k+1}),$$

where W_{k+1} is an l_k by l_{k+1} weight matrix and b_{k+1} is a l_{k+1}dimensional bias vector. Finally, the output of the autoencoder Y is calculated as

$$Y = H_h W_h + b_h,$$

where W_h is an l_{h-1} by l_h weight matrix and b_h is an l_h dimensional bias vector. The autoencoder learns the parameters in weight matrices $W_1, W_2, \ldots, W_h$ and bias vectors $b_1, b_2, \ldots, b_h$ by minimizing the mean squared error (MSE) between input X and output

Y on nonzero values of X. Let W be the set of weight matrices $W_1, W_2, \ldots, W_h$ and b be the set of bias vectors $b_1, b_2, \ldots, b_h$, then

$$MSE\ (W, b) = \frac{\sum_i^n \sum_j^m (Y_{ij} - X_{ij})^2 I(X_{ij} \neq 0)}{\sum_i^n \sum_j^m I(X_{ij} \neq 0)}.$$

The weight and bias parameters $(\widehat{W}, \hat{b})$ are given by

$$(\widehat{W}, \hat{b}) = argmin_{(W,b)} MSE\ (W, b),$$

where $I(x)$ is an indicator function that outputs one for nonzero input and zero otherwise. The MSE is the loss function in the optimization process.

The minimization of MSE is a non-convex optimization problem [37] and the backpropagation algorithm [102] is utilized for training the autoencoder (Methods). In the imputation step, the zero entries in the input matrix X are replaced by their nonzero counterparts in the output matrix Y. Let $\widehat{AE}$ be the autoencoder with parameters $(\widehat{W}, \hat{b})$ learned by backpropagation, then the imputed scRNA-seq data matrix $\hat{Y}$ is calculated as

$$\hat{Y} = X + \widehat{AE}(X) \circ I(X = 0),$$

where $\circ$ is the element-wise product.

Several modifications to the original model design have been made since the debut of autoencoder-based imputation methods. For example, DCA [32] models the scRNA-seq data by a negative binomial distribution with or without zero-inflation (NB or ZINB) and learns the autoencoder by maximizing the likelihood of NB or ZINB calculated by the output Y; scVI [31] learns a variational autoencoder [103] by forcing the hidden layers to follow a ZINB distribution; DeepImpute [104] learns the autoencoder by minimizing the weighted MSE between two sets of highly correlated genes in input and output; LATE [105] treats cells or genes as observations to learn two autoencoders and selects the one with smaller MSE; scScope [106] learns an iterative autoencoder by using the imputed data as input

repeatedly. Despite the aforementioned modifications, the essential structure, hyperparameters, and training process of the autoencoder for imputing scRNA-seq data remain the same.

4.2.2 Three masking schemes for introducing missing values

In scRNA-seq data, the knowledge of which observed zeros are truly missing is unknown due to the lack of external reference [22,33]. To evaluate the overall imputation accuracy in this scenario, we design three masking schemes that introduce missing values to scRNA-seq data and measure the difference between the imputed and true values on the masked data (Figure 1b-1d). These masking schemes represent different assumptions of missing mechanisms in scRNA-seq data [26].

First, we randomly mask 50% nonzero entries in the scRNA-seq data matrix (set their values to zero). We call this masking scheme random masking in the following text. Random masking indicates that the missing mechanism is completely independent of the true gene expression levels. It has been widely used in previous work to evaluate the imputation accuracy [22]. Second, we mask the nonzero entries less or equal to their median in the scRNA-seq data matrix. We call this masking scheme median masking in the following text. Median masking assumes a complete dependence of the missing mechanism on the true gene expression levels. Third, we assume that the probability of missing values for one gene depends on the mean expression level of that gene across cells. Lowly expressed genes are more likely to have missing values than highly expressed genes. Specifically, for gene i, let μ_i be the mean expression level of nonzero values across cells (log-transformed read count), and p_i be the probability of missing

values. Then the missing mechanism can be modeled by a double exponential function [55]

$$p_i = exp\ (-\lambda \mu_i^2),$$

where λ is a parameter learned from scRNA-seq data. Let Z_{ij} be a random variable that indicates whether to mask the nonzero expression of gene i in cell j, then $Z_{ij} \sim Bernoulli\ (p_i)$. All nonzero expression of gene i with $Z_{ij} = 0$ will be masked. The value of λ was determined such that 50% of nonzero entries in the scRNA-seq data matrix were masked. We call this masking scheme double exponential masking in the following text.

4.2.3 Impact of autoencoder architecture on the imputation accuracy

We collect 12 real scRNA-seq datasets to evaluate the overall imputation accuracy of the autoencoder under a variety of architectures. These datasets cover a wide range of cell types, sequencing depths, zero rates, and experimental platforms (Supplementary Table S1). We apply the three masking schemes to generate three sets of masked data, each containing 12 datasets. To make different datasets comparable, we use the normalized root MSE (NRMSE; Methods) and Pearson correlation coefficient between the imputed and true masked values to measure the imputation accuracy. We call them imputation NRMSE and imputation correlation, respectively, in the following text.

We build autoencoders of different architectures by increasing the number of hidden layers (depth) from 1 to 15. For each depth, we set the number of hidden units per layer (width) to 32, 64, 128, or 256, respectively. All hidden layers are fully connected with the same number of hidden units. Different depth-width combinations generate 60 (15×4) autoencoders in total. We choose the rectified linear unit (ReLU) [35] as the activation

function and train the autoencoders by the Adam optimization algorithm [107] (Methods). We set 10 random seeds in the training of each autoencoder-dataset combination to obtain 10 different imputed datasets. We average the corresponding imputation NRMSEs and imputation correlations to reduce the variability introduced by the stochastic training process.

Figures 2a and 2b show the impact of depth and width on the imputation NRMSE and correlation based on the random masking scheme. First, the deeper autoencoders provide lower imputation NRMSE and higher imputation correlation. The benefits of depth are more significant when the number of layers is less than 10. Second, the narrower autoencoders (32 hidden units per layer) typically provide more accurate imputation than wider autoencoders (64 or more hidden units per layer) of the same depth. This finding is consistent with the observation in computation vision study that deeper and narrower neural networks have better performance in multiple tasks (e.g., image classification and object detection) [108,109]. We observe a similar relationship between network architecture and imputation accuracy under the double exponential masking scheme, except for the dataset `bmmc` (Supplementary Figure S1). Conversely, the same relationship breaks under the median masking scheme, where deeper autoencoders mostly reduce the imputation accuracy while the width does not have a significant impact (Supplementary Figure S2). The imputation accuracy on the random masking is the highest among the three masking schemes, followed by the doublet exponential masking. The imputation fails on most datasets under median masking, indicated by many larger-than-one imputation NRMSEs and close-to-zero imputation correlations (Supplementary Figure S2).

The previous result is likely due to the different gene-to-gene associations preserved by different masking schemes. Both random masking and double exponential masking are stochastic processes. The difference between them is whether the probability of masking depends on the original gene expression levels. Under these two masking schemes, some strong singles (large values in the scRNA-seq data matrix) are masked while others are left, making the gene-to-gene association similar in masked and unmasked values. Therefore, the autoencoder is able to learn this common association from nonzero values to impute the masked values. Because of the high complexity of the scRNA-seq data [9], autoencoders with high representational capacity (deep) and low tendency toward overfitting (shallow) are more capable of learning this association and provide more accurate imputation.

On the other hand, median masking is a deterministic process that only masks the small nonzero values in scRNA-seq data (the medians of nonzero values in most real datasets in our analysis are one or two). This hinders the autoencoder from accurately imputing masking values for two reasons. First, very small values in scRNA-seq data are more likely to be random noise caused by the technical variance from the experimental process [110]. It is infeasible for autoencoders or any machine learning methods to recover random noise from singles in unmasked data. Second, for those small values that are actually weak singles instead of random noise, their gene-to-gene association is systematically different from the one of strong singles in unmasked data. This discrepancy causes the domain shift issue [111] and makes autoencoders trained on unmasked data generalize poorly on masked data, resulting in inaccurate imputation. Due to any of those two reasons, high model complexity (deep) causes overfitting and low imputation

accuracy, as shown in Supplementary Figure S2. The following analysis will mainly focus on random masking and double exponential masking.

4.2.4 Impact of activation function on the imputation accuracy

An activation function is a nonlinear transformation applied to the hidden units of the neural network [36]. It provides autoencoders the capacity to learn complex nonlinear patterns, e.g., the gene-to-gene association in scRNA-seq data. ReLU is a widely used activation function in autoencoder-based imputation methods, motivated by its success in computer vision study [36]. However, the validity of using ReLU for imputing scRNA-seq data has rarely been discussed, and the empirical comparison between ReLU and other activation functions is lacking.

Here, we train autoencoders with seven different activation functions, including sigmoid, tanh, ReLU, LeakyReLU (with two different hyperparameter settings) [112], ELU [113], and SELU [114], to compare their impact on the imputation accuracy (Methods). For each activation function, we imputed the aforementioned 12 scRNA-seq datasets using 20 autoencoders obtained by setting different random seeds in the training process. Figure 3 and Supplementary Figure S3 compare the distributions of imputation NRMSEs and imputation correlations for different activation functions under random masking or double exponential masking. We observe that sigmoid and tanh outperform other activation functions in all datasets under two masking schemes. Additionally, the variability of imputation NRMSE and imputation correlation are significantly lower for sigmoid and tanh than others, indicating more stable imputation accuracy. Between sigmoid and tanh, they have similar imputation accuracy except for datasets `pbmc` and

human_mix. The performance of sigmoid is more stable on datasets mbrain, pbmc, human_mix, and mouse_cortex.

The comparison of different activation functions has three insights. First, one argument for using ReLU is that it can improve the low dimensional representation of data by introducing sparsity into the hidden layers [112]. Contrarily, sigmoid and tanh (mostly) generate nonzero hidden units, and thus, dense hidden layers. Our empirical results show that the imputation of scRNA-seq data did not benefit much from sparse hidden layers, probably because the scRNA-seq data itself is highly sparse and thus needs a dense representation in the low-dimensional space (hidden layers). Second, Leaky ReLU, ELU, and SELU are modifications to ReLU by inducing small nonzero output values in hidden units when the input is negative (Methods). They generate pseudo-sparsity in the hidden layers and avoid the dead ReLU problem [115]. Also, the nonzero hidden units provide the autoencoder more flexibility to adjust its parameters. However, our empirical result shows no consistent improvement of those activation functions over ReLU in terms of imputation accuracy. A possible interpretation is that the derivative shape of the activation function is critical to the training of the autoencoder — sigmoid and tanh have continuous derivatives, while all ReLU-related activation functions have discrete derivatives. Third, we do not observe vanishing gradient or exploding gradient problems [116] in the training of autoencoders with sigmoid or tanh activation functions. We suspect that the appropriate preprocessing and normalization of scRNA-seq data stabilize the gradients in the training of the autoencoder.

4.2.5 Impact of regularization on the imputation accuracy

Regularization is a technique to constrain the complexity of machine learning models such that they can generalize better to the data not used in training [83]. There are two commonly used regularization methods among others to improve the imputation accuracy of autoencoders — weight decay [37] and dropout [38]. Weight decay incorporates the $L2$ norm of weight parameters into the loss function to penalize large weights in the autoencoder. The weight and bias parameters under the weight decay $(\widehat{W}, \hat{b})'$ are given by

$$(\widehat{W}, \hat{b})' = \ argmin_{(W,b)} MSE\ (W, b)\ + \lambda ||W||_2^2,$$

where $||W||_2$ is the $L2$ norm of weight parameters and λ is a tuning parameter that controls the degree of penalization.

Rather than penalizing the scale of weights, dropout regularization randomly sets a proportion of hidden units to zero in the training of autoencoders. It forces the autoencoder not to rely on particular hidden units and thus reduces overfitting [38]. Specifically, suppose that Z_k is a random vector with the same dimension as the hidden layer k. Each random variable in Z_k independently follows a Bernoulli distribution with parameter $p_k \in (0, 1)$. Then in the training, the calculation of hidden layer $k + 1$ under dropout regularization is

$$H_{k+1} = f\ [(H_k \circ Z_k)W_{k+1} + b_{k+1}],$$

where $\circ$ is the element-wise product. Note that the calculation of hidden layers in the testing (imputation) does not involve the dropout operation. In our analysis, we set $p_1 = p_2 = \ldots = p_h = p$, where h is the number of hidden layers. We call p dropout rate in the following text.

Some autoencoder-based imputation methods have utilized weight decay or dropout in their implementations [104,117]. However, the selection of regularization methods and corresponding hyperparameters (i.e., the λ in weight decay and the p in dropout) tuning are mostly ad hoc [24]. To examine the impact of regularization on the imputation accuracy, we train autoencoders with weight decay or dropout and impute the aforementioned 12 real scRNA-seq datasets (Methods). We set the hyperparameters to a broad range of values and compare their imputation NRMSE and imputation correlation (Figure 4 and 5; Supplementary Figure S4 and S5). All autoencoders have the same architecture and activation function — 10 fully connected hidden layers, 32 hidden units per hidden layer, and a sigmoid activation function. We set 10 random seeds in the training of each autoencoder-dataset combination to obtain 10 different imputed datasets. We average the corresponding imputation NRMSEs and imputation correlations to reduce the variability introduced by the stochastic training process.

Under the random masking, weight decay barely improves the imputation accuracy except for datasets `mouse_spleen` and `human_mix`. The larger values of λ even reduce the imputation accuracy, which indicates an over-regularization (Figure 4). On the other hand, dropout improves the imputation NRMSE on six datasets and imputation correlation on 11 datasets if paired with an appropriate dropout rate p. The optimal ps are small to moderate values (between 0.02 and 0.2; Figure 5) in terms of imputation NRMSE while moderate to large values (between 0.2 and 0.4) in terms of imputed correlation.

Under the double exponential masking, both regularization methods improve the imputation accuracy (Supplementary Figure S4 and S5). Weight decay improves the imputation NRMSE on six datasets and imputation correlation on 11 datasets

(Supplementary Figure S4). The optimal λs are small to moderate values (between 1e-7 and 5e-5) in terms of imputed NRMS while moderate to large values (between 5e-6 and 5e-4) in terms of imputed correlation. The dropout also exhibits a strong positive impact on imputation accuracy, with imputation NRMSE improved on 10 datasets and imputation correlation improved on all 12 datasets (Supplementary Figure S5). The optimal dropout rate p is similar to those under random masking, with smaller values in terms of imputation NRMSE and larger values in terms of imputation correlation.

The impact of regularization on imputation accuracy can be summarized in three perspectives. First, dropout is more capable of improving both imputation NRMSE and imputation correlation than weight decay. The autoencoders with dropout improve the imputation on more datasets under both masking schemes - accuracy measurement combinations. Second, both regularization methods are more effective under double exponential masking than under random masking. As illustrated in the previous section, double exponential masking tends to mask small nonzero values in the scRNA-seq data matrix and thus introduces slightly different gene-to-gene associations between masked and unmasked values. This difference causes moderate domain shifting and overfitting in the autoencoder. Instead, random masking does not have this issue. Therefore, as a counter-overfitting technique, the effect of regularization is stronger under double exponential masking where more overfitting exists. Third, the optimal hyperparameters of regularizations largely depend on the datasets and masking schemes. It is difficult to find a universal hyperparameter setting that accommodates all scenarios. Interestingly, compared with the imputation NRMSE, the imputed correlation requires a higher degree of regularization to achieve its optimum under both weight decay and dropout.

4.2.6 Impact of autoencoder design on cell clustering

The ultimate goal of imputation is to improve the downstream bioinformatic analysis through the enhancement of signals in the sparse scRNA-seq data [5]. We collect 20 real scRNA-seq datasets with ground-truth cell types to examine the impact of autoencoder design on cell clustering (Supplementary Table S2). The datasets used in cell clustering are different from those in the evaluation of overall imputation accuracy. Specifically, we first conduct k-means clustering on the pre-imputed datasets and calculate the adjusted Rand index (ARI) and adjusted mutual information (AMI) to measure the clustering performance (Methods). We call them baseline ARI and baseline AMI in the following text. Second, we train autoencoders with various architectures, activation functions, and regularizations to impute the aforementioned 20 datasets. Finally, we conduct k-means clustering on each imputed dataset and calculate the corresponding ARI and AMI. We call them imputation ARI and imputation AMI in the following text.

Figure 6a, Supplementary Figure S6a, and S7a show the impact of autoencoder architecture on cell clustering. Similar to the previous analysis, we increase the depth of autoencoders from 1 to 15 and set the width to 32, 64, 128, and 256 for each depth, respectively. All hidden layers in each autoencoder are fully connected with the same number of hidden units. Different depth-width combinations generate 60 (15×4) autoencoders in total. We chose sigmoid as the activation function because of their superior performance in the evaluation of imputation accuracy. We set five random seeds in the training of each autoencoder-dataset combination to obtain five different imputed datasets. We average the corresponding imputation ARIs and imputation AMIs to reduce the variability introduced by the stochastic training process. We observe that deeper

autoencoders provide a greater improvement on cell clustering than their shallower counterparts. The benefit of depth saturates after the number of hidden layers is greater than 10. On the other hand, width has no significant impact on cell clustering (Figure 6a; Supplementary Figure S6a; Supplementary Figure S7a). Those patterns are consistent in terms of both imputed ARI and imputed AMI. Surprisingly, the cell clustering after imputation only outperforms baseline ARIs on eight datasets and baseline AMIs on four datasets, regardless of autoencoder architectures.

Figure 6b, Supplementary Figure S6b, and S7b compare the impact of activation functions on cell clustering. Similar to the previous analysis, we train autoencoders with seven different activation functions, including sigmoid, tanh, ReLU, LeakyReLU (with two different hyperparameters), ELU, and SELU. For each activation function, we impute the aforementioned 20 scRNA-seq datasets using 10 autoencoders obtained by setting 10 different random seeds in the training process. All autoencoders have 10 fully connected hidden layers with 32 hidden units per layer. We observe that sigmoid and tanh outperform other activation functions in terms of both imputed AMI and imputed ARI on all datasets. They also exhibit more stable cell clustering performance than other activation functions. The performance of sigmoid and tanh are fairly comparable except for datasets `Zeisel` and `klein`, where tanh has a slight advantage over sigmoid.

Figure 6c-6d, Supplementary Figure S6c-S6d, and S7c-S7d show the impact of regularization on cell clustering. Similar to the previous analysis, we utilize weight decay or dropout in autoencoders and adjust their hyperparameters in a broad range of values. All autoencoders have 10 fully connected hidden layers with 32 hidden units per layer and sigmoid activation functions. Interestingly, weight decay significantly improves cell

clustering — cell clustering after imputation with weight decay outperforms baseline ARIs on all 20 datasets and baseline AMIs on 18 datasets if paired with appropriate hyperparameter. The optimal hyperparameters of weight decay are mostly between 0.01 and 0.1 on both imputed ARI and imputed AMI. However, the same improvement does not happen on dropout regularization — cell clustering after imputation with dropout only outperforms baseline ARIs on eight datasets and baseline AMIs on four datasets, if paired with appropriate dropout rate p. Dropout regularization has limited benefits for autoencoders to improve cell clustering through imputation. Its optimal hyperparameters cover a broad range depending on the datasets.

The previous comparison demonstrates the critical role of regularization for autoencoders to improve cell clustering. Autoencoder-based imputation methods impute all zero values in the scRNA-seq data matrix without differentiating between biological zeros from technical zeros. This causes an over-imputation issue because only technical zeros need to be imputed [118]. Over-imputation introduces false signals to scRNA-seq data. If the harms of false signals surpass the benefits of true signals recovered, then overall, the imputation cannot improve cell clustering, as shown in Figure 6, Supplementary Figure S6, and S7. The weight decay regularization relieves over-imputation by shrinking the weights of the autoencoder while still being able to recover true signals at the same time. Therefore, weight decay with appropriate degrees of penalization improves cell clustering. Contrarily, dropout regularization tries to break the reliance on particular hidden units instead of adjusting the size of weight parameters. Our empirical result shows that this regularization mechanism is not able to effectively reduce over-imputation or improve cell clustering. Note that there is no over-imputation issue in the evaluation of

overall imputation accuracy because the technical zeros (i.e., masked values) are known in advance and imputed NRMSE and imputed correlation are calculated based on those values.

4.2.7 Impact of autoencoder design on DE gene analysis

The enhancement of signals in scRNA-seq data by imputation is supposed to benefit another important downstream analysis — the identification of differentially expressed (DE) genes. To examine the impact of autoencoder design on DE gene analysis, we utilize simulator scDesign [51] to generate 20 synthetic datasets with ground truth DE genes (Methods). Each synthetic dataset is generated by learning the distribution of gene expression in one real scRNA-seq dataset (20 real datasets in total; Supplementary Table S3). These real datasets (and their synthetic counterparts) cover a wide range of biological and technical conditions. We use synthetic data in this analysis since the ground truth DE genes are typically unknown in real scRNA-seq datasets.

After simulation, we apply the MAST method [56] to pre-imputed synthetic datasets to identify DE genes and calculate the corresponding precision, recall, and true negative rate (TNR). We call them baseline precision, baseline recall, and baseline TNR in the following text. Next, we train autoencoders with various architectures, activation functions, and regularizations, and impute the aforementioned 20 synthetic datasets. Finally, we apply MAST to each imputed dataset and calculate the corresponding precision, recall, and TNR (Methods). We call them imputed precision, imputed recall, and imputed TNR in the following text.

Figure 7a, Supplementary Figure S8a, S9a, and S10a show the impact of autoencoder architecture on imputed precision, imputed recall, and imputed TNR. The settings of depth,

width and activation function for autoencoders are the same as in the evaluation of cell clustering (1 to 15 fully-connected hidden layers; 32, 64, 128, or 256 hidden units per layer; sigmoid activation function). We set five random seeds in the training of each autoencoder-dataset combination to obtain five different imputed datasets. We average the corresponding imputed precisions, imputed recalls, and imputed TNR to reduce the variability introduced by the stochastic training process. First, we observe that the imputed precision is similar across different depths and widths of the autoencoder, except for the dataset `Interneurons`, `Epithelial_cells`, and `astrocytes`, where deeper autoencoders slightly improved the imputed precision. Overall, the imputation improves the precision of identified DE genes over the baseline on 19 datasets. Second, deeper autoencoders provide higher imputed recall while the benefits generally saturate after the depth passes five. On the other hand, width has no significant impact on the imputed recall. Unfortunately, the imputation fails to improve the imputed recall over the baseline on all datasets, regardless of the autoencoder architecture. Third, the impact of depth and width on imputed TNR is limited, partially because all baseline TNRs are already close to one, and the gaps between them and imputed TNRs are less than 0.05. Finally, it is worth noting that the imputation by autoencoders has a clear priority to improving the precision over the recall in DE gene analysis. The increase of imputed precision often comes at the cost of the decrease of imputed recall. In other words, the imputation by the autoencoder makes the DE gene analysis more conservative.

Figure 7b, Supplementary Figure S8b, S9b, and S10b compare the impact of activation functions on the imputed precision, imputed recall, and imputed TNR. Again, we train autoencoders with seven different activation functions, and for each of them, we

impute the aforementioned 20 synthetic datasets using 10 autoencoders obtained by setting different random seeds in the training process. All autoencoders have 10 fully connected hidden layers with 32 hidden units per layer. In terms of imputed precision, sigmoid and tanh outperform other activation functions on nine datasets and had comparable performance to others on 10 datasets. The only exception is the dataset `Endotherial_cell`, where ELU and SELU achieve the highest imputed precision. The advantage of sigmoid and tanh is more obvious in terms of imputed recall — they outperform other activation functions on 15 datasets. The comparison among activation functions on the imputed TNR is similar. The performance of sigmoid and tanh is generally comparable and they both provide a more stable improvement on the identification of DE genes than other activation functions.

Figure 7c-7d, Supplementary Figure S8c-S8d, S9c-S9d, and S10c-S10d show the impact of regularizations on the imputed precision, imputed recall, and imputed TNR. Again, we add weight decay or dropout to autoencoders and adjust their hyperparameters as in the previous analysis. All autoencoders have 10 fully connected hidden layers with 32 hidden units per layer and sigmoid activation functions. We observe that weight decay exhibits greater improvement on all three measurements of DE gene analysis than dropout. Specifically, weight decay improves the imputed precision, imputed recall, and imputed TNR over their no-regularization counterparts on 9, 20, and 20 datasets, respectively, if paired with appropriate hyperparameters. Moreover, weight decay with optimal hyperparameters makes imputed recall and imputed TNR surpass the baseline in all datasets. Dropout is able to improve imputed precision, imputed recall, and imputed TNR over their no-regularization counterparts on 6, 17, and 17 datasets, respectively, if

paired with appropriate hyperparameters. However, the improvement is much less than weight decay and fails to make the imputed recall and imputed TNR surpass the baseline on all but one dataset.

Similar to the evaluation of cell clustering, regularization is also critical for autoencoders to improve DE gene analysis. Again, the weight decay reduces over-imputation by constraining the size of weight parameters in autoencoders. With an appropriate degree of penalty, weight decay balances the recovery of true signals against the introduction of false signals to scRNA-seq data. Our empirical result also shows that dropout does not reduce over-imputation well as weight decay. Finally, hyperparameter tuning plays a critical role in determining the impact of regularization on DE gene analysis, especially for weight decay. The optimal hyperparameters of weight decay are mostly between 0.1 and 1 in terms of imputed recall and imputed TNR while less than 0.01 in terms of imputed precision. On the other hand, the DE analysis is relatively insensitive to the hyperparameter of dropout — the imputed precision, imputed recall, and imputed TNR are similar across a wide range of dropout rates.

4.3 Discussion

The high degree of sparsity is one of the major hurdles of analyzing scRNA-seq data, especially those induced by technical variability. This issue has brought much attention in scRNA-seq fields since the emergence of the first experimental protocols of scRNA-seq [119]. There are more than 70 computational imputation methods that explicitly or implicitly try to impute or denoise sparse scRNA-seq data [16]. The autoencoder-based imputation methods are motivated by the success of deep learning and its application to

signal recovery in computer vision [120]. Compared with traditional statistical and machine learning methods, they exhibit several advantages. First, autoencoder-based imputation methods require no assumptions about the underlying distribution of scRNA-seq data. This data-driven characteristic avoids model specification error and bias in traditional methods. Second, autoencoder-based imputation methods can effectively handle large-scare scRNA-seq data by using innovative hardware, e.g., the GPU. Third, autoencoder-based imputation methods have high flexibility due to their neural network work design. They can incorporate multiple functionalities in one framework, including imputation, dimension reduction, and batch effect normalization [31].

Albeit all aforementioned advantages, however, how to design autoencoders remains a great challenge due to the large number of hyperparameters. The successful applications of autoencoders in other fields, especially computer vision, rely on systematic empirical studies conducted on massive datasets to search for the best hyperparameters. Current imputation methods mainly borrow the experience from those fields to set up their autoencoders. Although some of those practices may also perform well on scRNA-seq data, there is no guarantee that every design consideration is exactly the same among distinctive data types. Our comprehensive empirical study echoes the previous argument. On the one hand, the better performance of deeper and narrower autoencoders is consistent with the theoretical and empirical results widely accepted by the deep learning community. On the other hand, the observation that sigmoid and tanh outperform other activation functions, especially ReLU, is unexpected. This result reflects the unique characteristic of scRNA-seq data compared with image data. Our result is partially validated by another study which found that a neural network with tanh activation function

outperforms its ReLU counterpart in the cell-type classification task based on scRNA-seq data [121].

The shallow and wide autoencoder design in current methods may be motivated by the observation that deeper neural networks do not improve cell-type classification on scRNA-seq data [122]. The reason is that the ground-truth cell type labels are mainly generated based on known marker genes, which makes different cell types relatively easy to be separated and limits the classification capacity of deep neural networks. However, the imputation task is essentially a regression rather than a classification problem. The predictive variable is continuous gene expression values (after pre-processing) instead of discrete cell-type labels. Therefore, the imputation is a more difficult prediction task than cell-type classification and needs deep neural networks with high predictive capacity [37]. On the other hand, even though deeper autoencoders exhibit advantages in our study, the benefits saturate when the number of layers passes 10. This is a much shallower architecture compared to state-of-the-art deep neural networks with hundreds of layers in computer vision study. Still, it reflects the consistency between data complexity and model capacity. One image is typically saved in a three-dimensional tensor format (three RGB channels, width, and length) [123], which is much more complex than one cell in a one-dimensional vector format in scRNA-seq data. Any learning task on such complex data requires highly capable models (i.e., deeper neural networks).

We find that dropout improves more on the overall imputation accuracy while weight decay excels in downstream cell clustering and DE gene analysis. Although dropout does not directly penalize the size of weight parameters in the autoencoder, it actually introduces sparsity into the weight parameters by randomly shutting down connections

between hidden units. In other words, dropout can be understood as a stochastic $L1$ penalty. From this point of view, we actually observe that $L1$penalization benefits the overall imputation accuracy while $L2$ improves downstream analysis. We should also note that the three masking schemes are imitations of but not the true missing mechanism and ultimately, the goal of imputation is to enhance downstream analysis. Therefore, $L2$ penalization (weight decay) may provide stronger benefits in real-world applications.

In summary, the performance of autoencoder-based imputation methods is sensitive to key aspects of the autoencoder design, including architecture, activation function, and regularization. Borrowing practice learned from other fields does not guarantee optimal performance on scRNA-seq data. The future methodological development should pay more attention to those design aspects and also offer the flexibility that allows users to adjust them based on their specific applications.

4.4 Methods

4.4.1 Data preprocessing and normalization

All real and synthetic scRNA-seq datasets used in this study are count matrices. They are preprocessed and normalized by the following three steps. First, we remove genes expressed in less than three cells and cells with less than 200 genes expressed. Second, the gene expression counts of each cell are divided by the total counts of that cell (library size) and then multiplied by 10000 (library size normalization). The results are further added by one and then natural-log transformed. Third, we select 2000 highly variable genes by using the `vst` method implemented in the `FindVariableFeatures` function

of the `Seurat` package [124] (v 4.0). After preprocessing and normalization, the dimension of all scRNA-seq data matrices is cell number × 2000. Note that the previous preprocessing and normalization only apply to the pre-imputed datasets. The imputed datasets as the input of cell clustering and DE gene analysis will not go through the same process.

4.4.2 Training of autoencoders and imputation

All the training of autoencoders was implemented by the `Pytorch` deep learning library [125] (v 1.8.1) on a server with two Intel Xeon E5-2687W v4 CPUs, 256GB memory, an Nvidia Geforce RTX 2080 Ti GPU, and Ubuntu 18.04 system. After preprocessing, normalization, and masking (masking is only necessary for the evaluation of overall imputation accuracy), we split each dataset's 80% cells into a training set and another 20% cells into a validation set. We utilize the Adam optimization algorithm [107] to train the autoencoder set with a 0.001 learning rate and a 64 batch size. After every epoch of training on the training set, we impute the validation set using the current autoencoder and calculate the MSE between imputed and original nonzero values of the validation set. We stop the training until the aforementioned MSE does not decrease over 20 epochs or the total number of epochs surpasses 10000. In the imputation step, the trained autoencoder accepts the preprocessed and normalized scRNA-seq data matrix as input (with the dimension as cell number × 2000) and outputs a data matrix of the same dimension. The final imputed data matrix is generated by replacing the zero entries in the input matrix with their counterparts in the output matrix. The nonzero entries in the input matrix remain the same in the final imputed data matrix.

4.4.3 Calculation of imputation normalized root MSE (NRMSE)

Suppose X is the sparse scRNA-seq input matrix after appropriate preprocessing and normalization; $\hat{Y}$ is the imputed scRNA-seq data matrix; M is the set of masked entries in the scRNA-seq data matrix. The MSE between imputed and true masked values MSE_{mask} is calculated as

$$MSE_{\text{mask}} = \frac{\sum_i^n \sum_j^m (X_{ij} - \widehat{Y_{ij}})^2 I(X_{ij} \in M)}{\sum_i^n \sum_j^m I(X_{ij} \in M)}.$$

Suppose the mean masked values $\overline{X}_{\text{mask}}$ is

$$\overline{X}_{\text{mask}} = \frac{\sum_{i=1}^n \sum_{j=1}^m X_{ij} I(X_{ij} \in M)}{\sum_{i=1}^n \sum_{j=1}^m I(X_{ij} \in M)},$$

then the imputation NRMSE $NRMSE_{\text{imputation}}$ is calculated as

$$NRMSE_{\text{imputation}} = \frac{\sqrt{\text{MSE}_{\text{mask}}}}{\overline{X}_{\text{mask}}}.$$

4.4.4 Activation functions

We evaluate seven activation functions in this study, including logistic function (sigmoid), hyperbolic tangent function (tanh), rectified linear unit (ReLU), leaky ReLU (with two different hyperparameters), exponential linear units (ELU), and scaled exponential linear units (SELU). In neural networks, they accept a linear transformation of the outputs from the last layer as input and apply a nonlinear transformation on top of them. The shapes of the seven activation functions are shown in Supplementary Figure S11.

The sigmoid activation function is a bounded differentiable function with positive and continuous derivatives. It ranges from 0 to 1. The function form of sigmoid is given by

$$f(x) = \frac{1}{(1+e^{-x})}.$$

The tanh activation function is also a bounded differentiable function with positive and continuous derivatives. It ranges from -1 to 1. The function form of tanh is given by

$$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}.$$

The ReLU activation function conducts a threshold operation that outputs zeros for negative inputs and preserves the positive inputs. It ranges from 0 to $+\infty$ and has discrete derivatives. The function form of ReLU is given by

$$f(x) = \begin{cases} x & if \ x \geq 0 \\ 0 & if \ x < 0 \end{cases}.$$

The leaky ReLU activation function modifies ReLU by introducing a small negative slope when the input is negative. It ranges from $-\infty$ to $+\infty$ and has discrete derivatives. The function form of leaky ReLU is given by

$$f(x) = \begin{cases} x & if \ x \geq 0 \\ \alpha x & if \ x < 0 \end{cases},$$

where α is a hyperparameter. In our analysis, we set α to 0.01 and 0.2 — the default values in two popular deep learning libraries `Pytorch` [125] and `TensorFlow` [126].

The ELU activation function replaces the linear negative part of leaky ReLU with an exponential function. It ranges from $-\infty$ to $+\infty$ and has discrete derivatives. The function form of ELU is given by

$$f(x) = \begin{cases} x & if \ x \geq 0 \\ \alpha e^{x} - 1 & if \ x < 0 \end{cases},$$

where α is a hyperparameter. In our analysis, we set α to 1 — the default value in `Pytorch`.

The SELU activation function further adds a scale factor to ELU and changes its constant in the negative part. It ranges from $-\infty$ to $+\infty$ and has discrete derivatives. The function form of ELU is given by

$$f(x) = \tau \begin{cases} x & if\ x \geq 0 \\ \alpha e^x - \alpha & if\ x < 0 \end{cases},$$

where τ and α are predefined parameters with $\tau = 1.05$ and $\alpha = 1.67$.

4.4.5 Cell clustering analysis

We utilize the function `kmeans` in R programming language to conduct k-means clustering on the pre-imputed and imputed scRNA-seq datasets (Supplementary Table S2). We set parameter `centers` (i.e., k in the k-means clustering) to the correct number of cell types in each dataset. We set parameter `nstart` to 25, which repeats the clustering 25 times by randomly selecting 25 sets of initial cluster centers and returns the result with a minimum sum of pairwise distances within clusters [127]. The dimension of input data matrices for k-means clustering is cell number $\times$ 2000 without further dimension reduction. Note that before clustering, pre-imputed datasets are preprocessed by following the procedure described in the section "Data preprocessing and normalization." On the other hand, the imputed datasets are directly clustered by k-means clustering.

We use adjusted Rand index (ARI) and adjusted mutual information (AMI) to measure the performance of cell clustering. Let $U = \{u_1, u_2, \ldots, u_c\}$ be the true partition of c classes and $V = \{v_1, v_2, \ldots, v_c\}$ be the partition obtained by k-means clustering. Let n_i and n_j be

the numbers of observations in class u_i and cluster v_i, respectively. Let n_{ij} be the number of observations in both class u_i and cluster v_i. The ARI is calculated as

$$\frac{\sum_{i=1}^{c}\sum_{j=1}^{c}\binom{n_{ij}}{2}-\left[\sum_{i=1}^{c}\binom{n_i}{2}\sum_{j=1}^{c}\binom{n_j}{2}\right]\Big/\binom{n}{2}}{\left[\sum_{i=1}^{c}\binom{n_i}{2}+\sum_{j=1}^{c}\binom{n_j}{2}\right]\Big/2-\left[\sum_{i=1}^{c}\binom{n_i}{2}\sum_{j=1}^{c}\binom{n_j}{2}\right]\Big/\binom{n}{2}},$$

where n is the number of observations and $n=\sum_{i=1}^{c} \quad n_i=\sum_{j=1}^{c} \quad n_j$. The AMI is calculated as

$$\frac{2I(U,V)}{H(U)+H(V)},$$

where $I(U,V)$ is the mutual information of U and V, and $H(U)$ and $H(V)$ are the entropies of U and V respectively [128]. We utilize the functions `ARI` and `AMI` in the package `aricode` of R programming language to calculate ARI and AMI, respectively.

4.4.6 Simulation of synthetic scRNA-seq data

We utilize simulator scDesign [51] to generate 20 synthetic scRNA-seq data with ground-truth DE genes. 20 real datasets (Supplementary Table S3) are preprocessed by following the procedure described in the section “Data preprocessing and normalization.” For each real dataset, we execute function `design_data` in R package `scDesign` to simulate one synthetic dataset based on the distribution of gene expression in that real dataset. Each synthetic dataset contains two cell types with 1000 cells per type. 10% genes are differentially expressed between the two cell types in the synthetic dataset. The sequencing depth of each synthetic dataset is equal to the sequencing depth of the

corresponding real dataset (sequencing depth = library size × cell number). Other parameters of function `design_data` are set as their default values. All synthetic datasets are count matrices with dimensions as cell number × 2000.

4.4.7 DE gene analysis

We conduct DE gene analysis on the aforementioned 20 synthetic datasets and their imputed counterparts. For pre-imputed synthetic datasets, the gene expression counts of each cell are divided by the total counts of that cell (library size) and then multiplied by 10000 (library size normalization). The results are further added by one and then natural-log transformed. We utilize the function `FindMarkers` in R package `Seurat` to identify the DE genes between the two cell types. We set the parameter `test.use` to "`MAST`" and identify genes with Bonferroni-corrected p-values under 0.05 as DE genes. Based on the ground-truth DE genes, we calculate the precision, recall, and TNR for each pre-imputed synthetic dataset and imputed dataset.

4.5 Figures

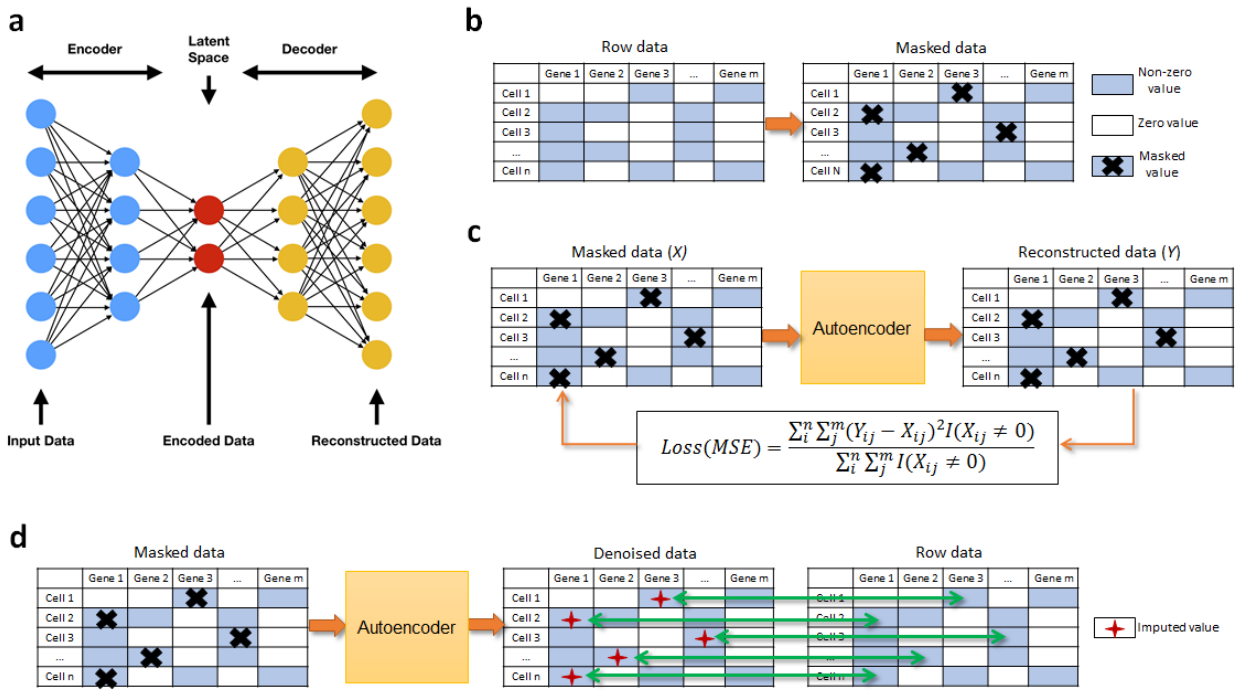


Figure 1. **Autoencoder and the measurement of imputation accuracy**. **a**, The basic structure of an autoencoder. **b**, The introduction of technical zeros by using three masking schemes. **c**, The training of autoencoders for imputation. **d**, the calculator of imputation accuracy on masked values.

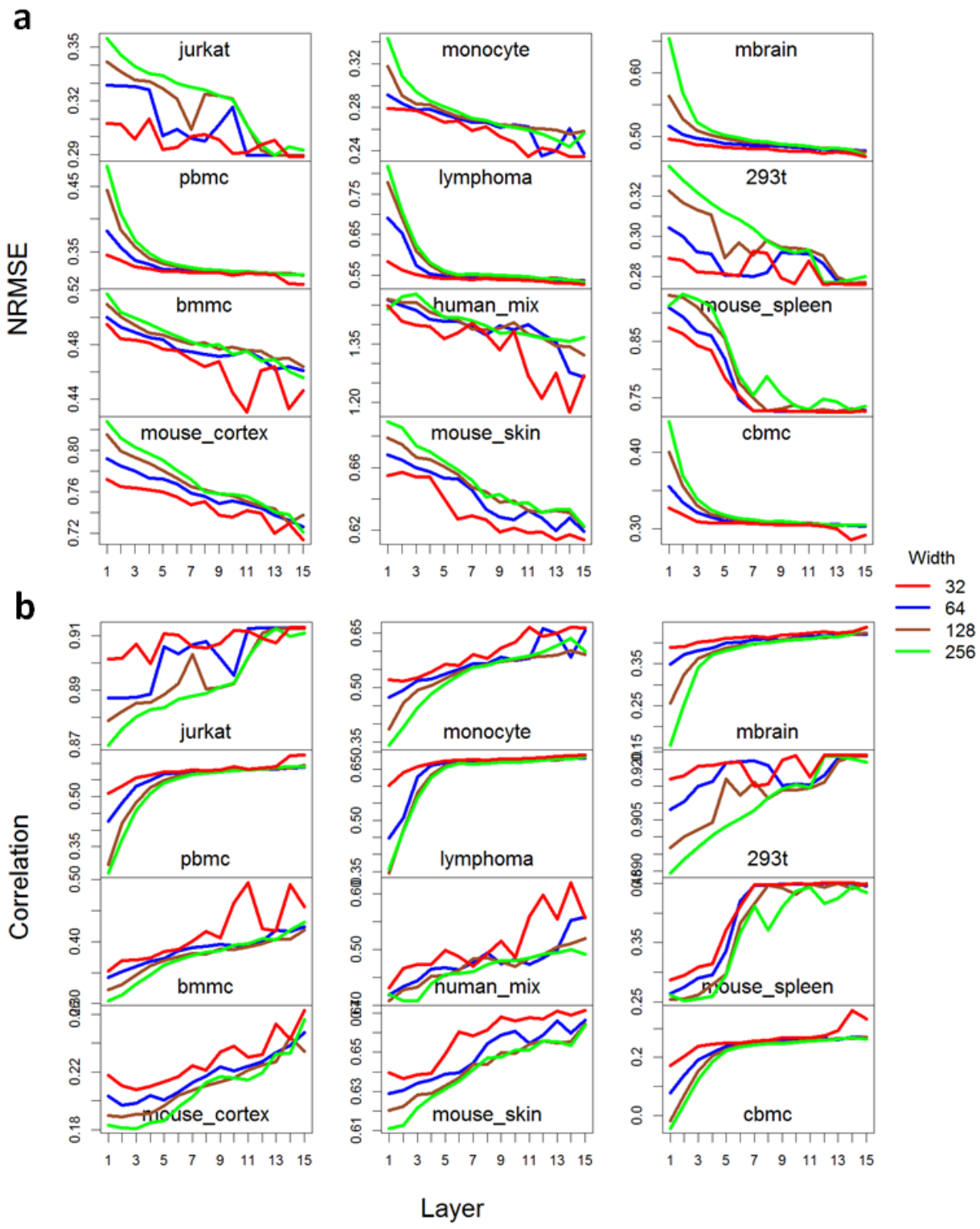


Figure 2. The impact of depth and width on the imputation NRMSE (**a**) and correlation (**b**) based on the random masking scheme.

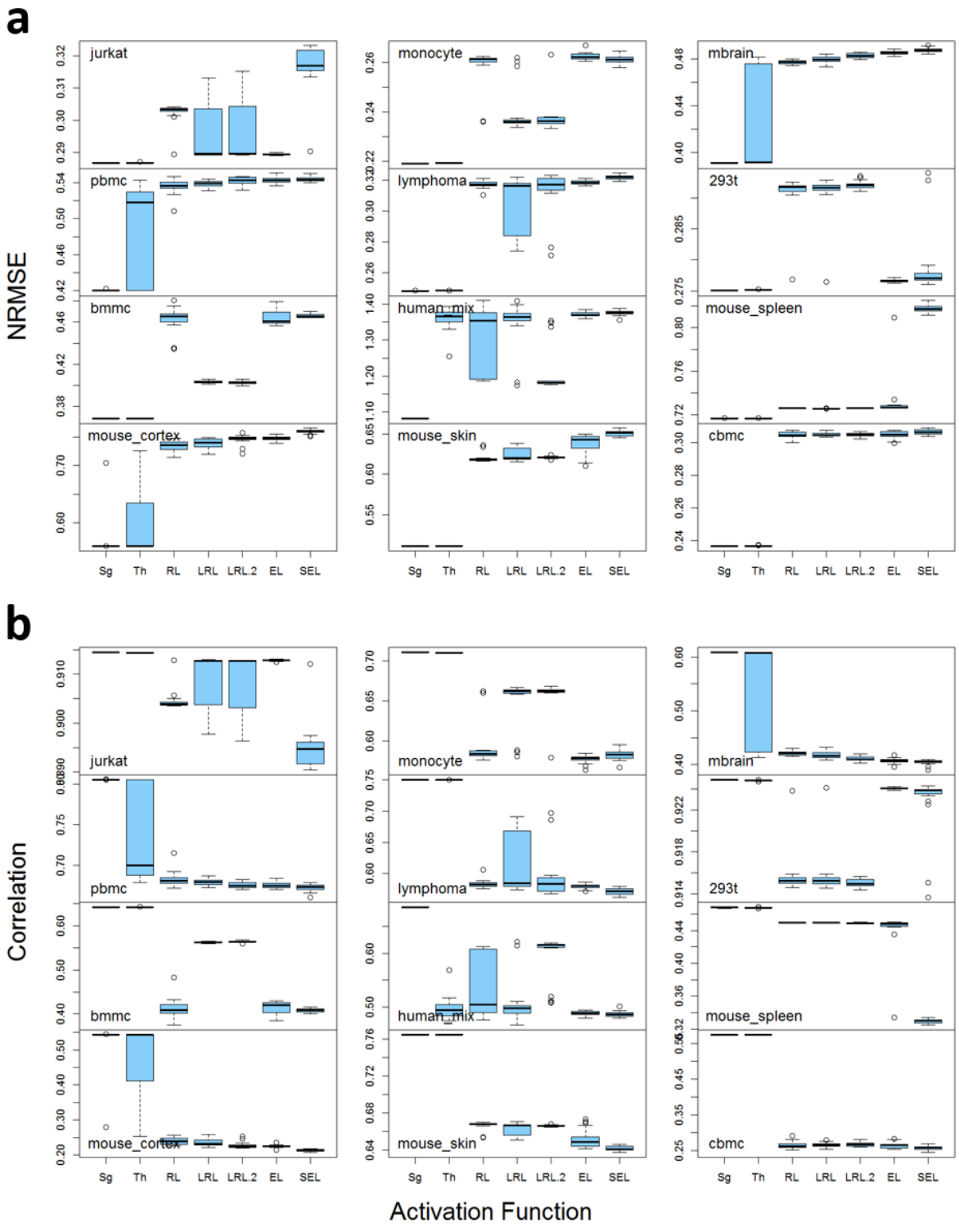


Figure 3. The impact of activation functions on the imputation NRMSE (**a**) and correlation (**b**) based on the random masking scheme. Sg: sigmoid; Th: tanh; RL: ReLU; LRL: LeakyReLU ($\alpha = 0.01$); LRL: LeakyReLU.2 ($\alpha = 0.2$); EL: ELU; SEL: SELU.

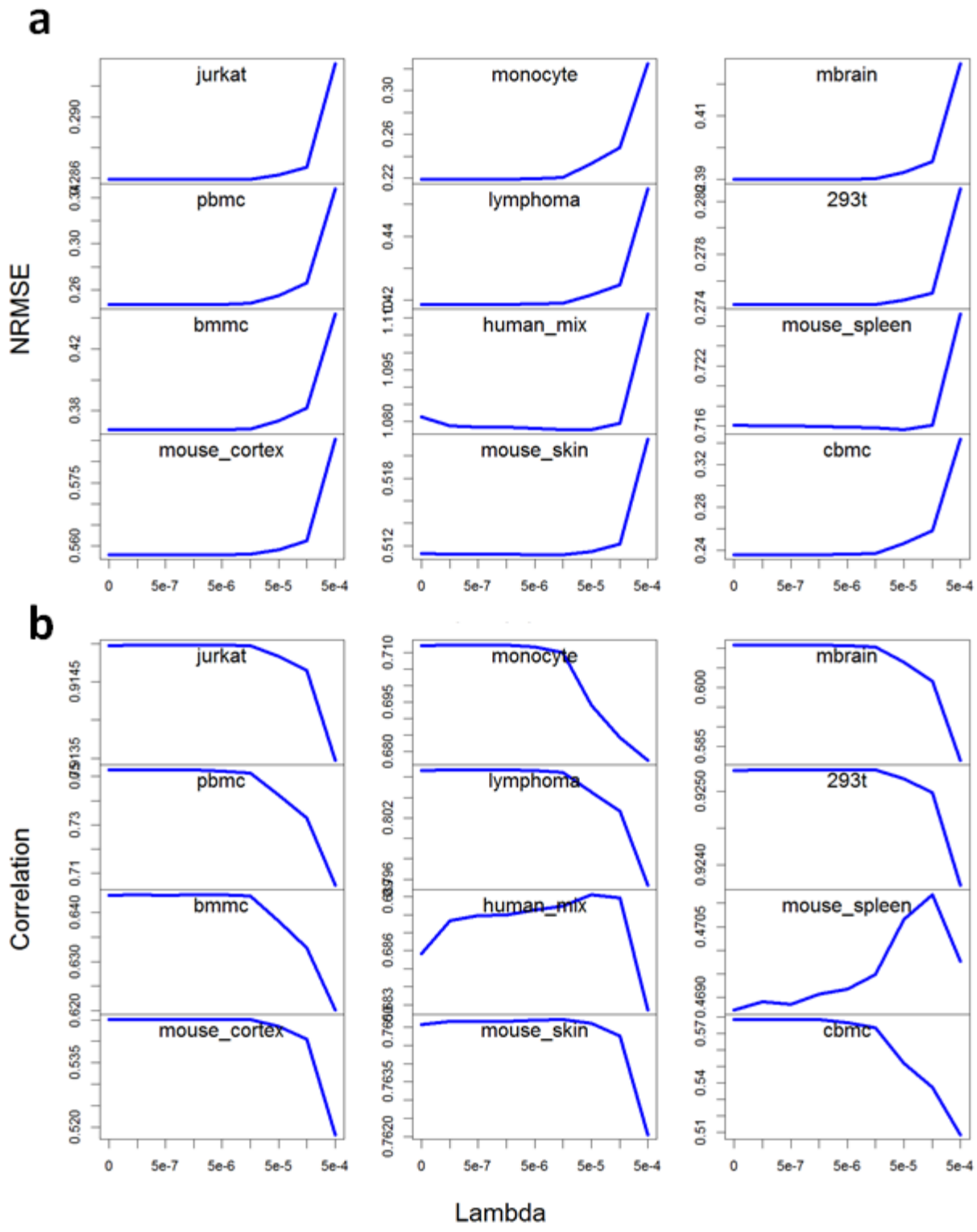


Figure 4. The impact of weight decay regularization on the imputation NRMSE (**a**) and correlation (**b**) based on the random masking scheme.

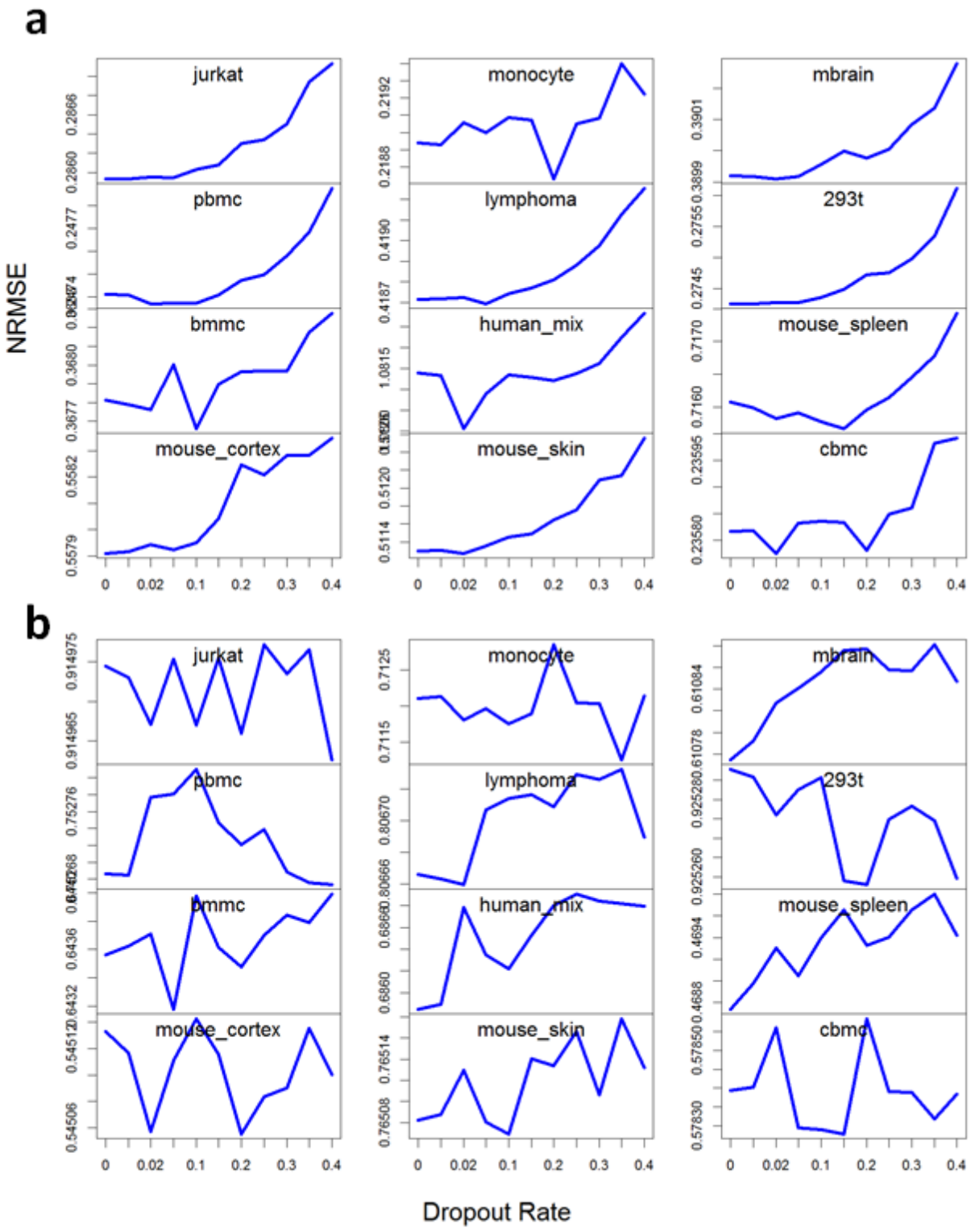


Figure 5. The impact of dropout regularization on the imputation NRMSE (**a**) and correlation (**b**) based on the random masking scheme.

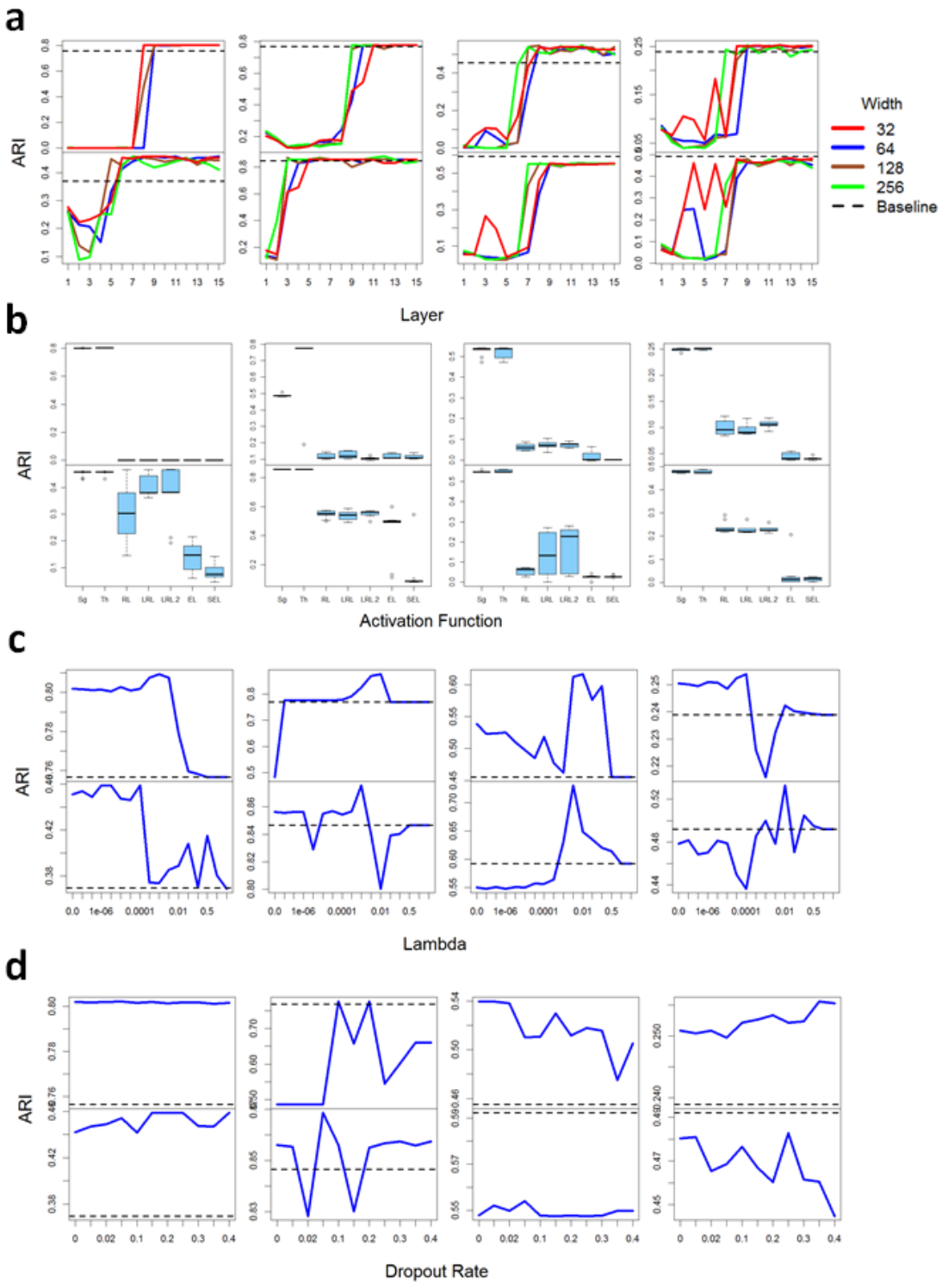


Figure 6. The impact of autoencoder design on the cell clustering measured by ARI. **a,** Depth and width. **b**, Activation function. **c**, Weight decay regularization. **d**, Dropout regularization. Eight datasets are shown here (from left to right; from top to bottom): Zhengmix4uneq, Zeisel, mouse1_umifm_counts, Silver, lake, li, human2_umifm_counts, human4_umifm_counts.

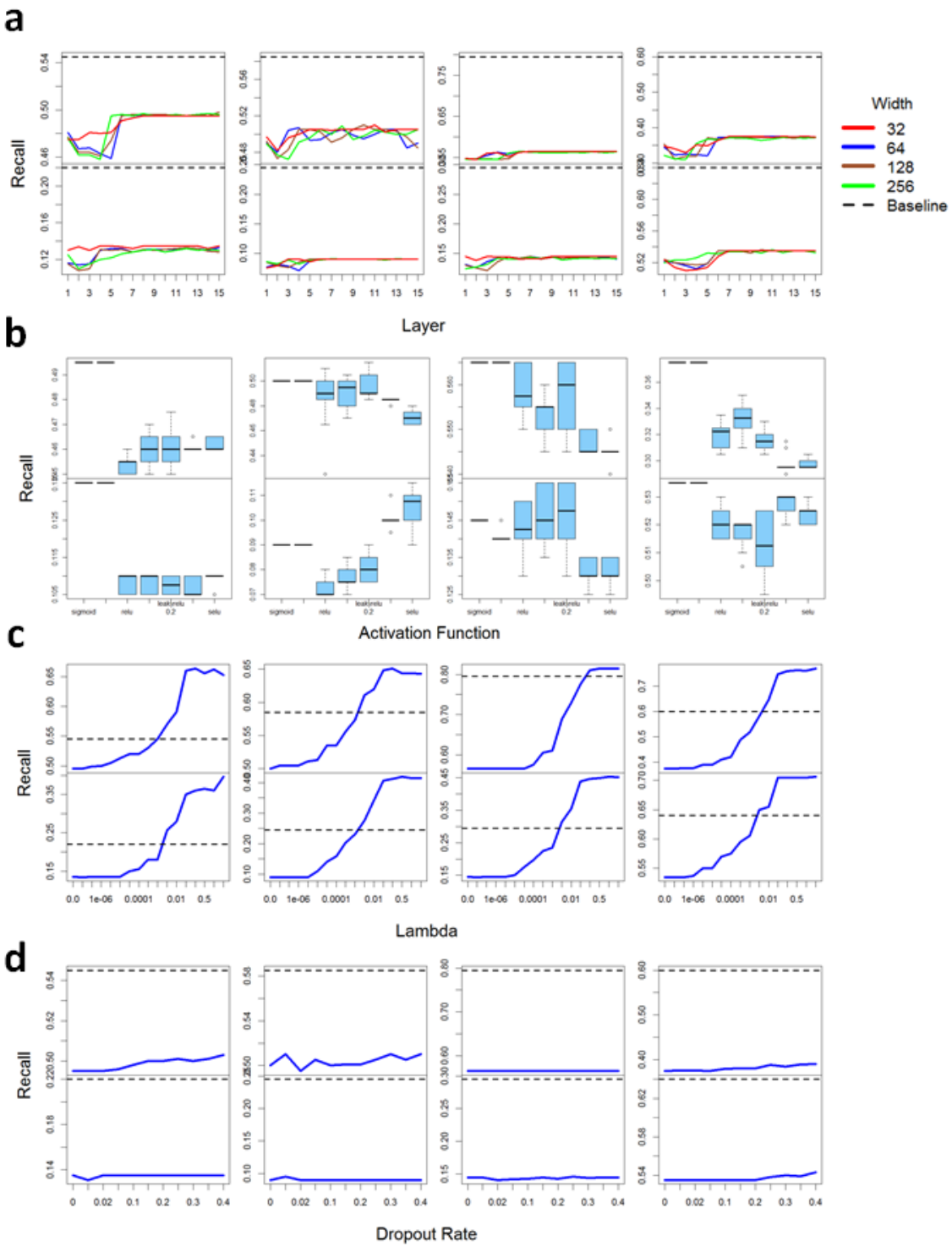


Figure 7. The impact of autoencoder design on the DE gene analysis measured by recall. **a,** Depth and width. **b**, Activation function. **c**, Weight decay regularization. **d**, Dropout regularization. Eight datasets are shown here (from left to right; from top to bottom): T, cd8, 293t, Fibroblasts, Macrophages, Endothelial_cells, Hematopoietic_stem_cells, NK.

4.6 Supplementary Figures and Tables

Supplementary Table S1. The 12 scRNA-seq datasets used to evaluate the overall imputation accuracy.

Dataset	Tissue/Cell type	Experimental protocol	# of cells	# of genes	Zero rate	Reference
jurkat	Jurkat cells	10x Genomics	3258	32738	90.23%	[52]
monocyte	CD14+ Monocytes	10x Genomics	2612	32738	98.59%	[52]
mbrain	Mouse brain cells	10x Genomics	9099	27998	90.97%	[52]
pbmc	Peripheral blood mononuclear cells	10x Genomics	7783	32738	97.89%	[52]
lymphoma	Lymphoma cells	10x Genomics	8412	33555	96.17%	[52]
293t	293T Cells	10x Genomics	2885	32738	89.56%	[52]
bmmc	Primary bone marrow mononuclear Cells	10x Genomics	1985	32738	97.64%	[52]
human_mix	Mixture of HEK293T and MCF7	Smart-seq-total	633	58660	89.58%	[129]
mouse_spleen	T-cells from the mouse spleen and small intestine	Smart-seq2	574	23998	84.79%	[130]
mouse_cortex	Mouse cortex and hippocampus	Fluidigm C1	3005	19972	81.21%	[131,132]
mouse_skin	Mouse skin	Fluidigm C1	1422	26024	90.03%	[133]
cbmc	Cord blood mononuclear cells	CITE-seq	8617	14438	95.54%	[134]

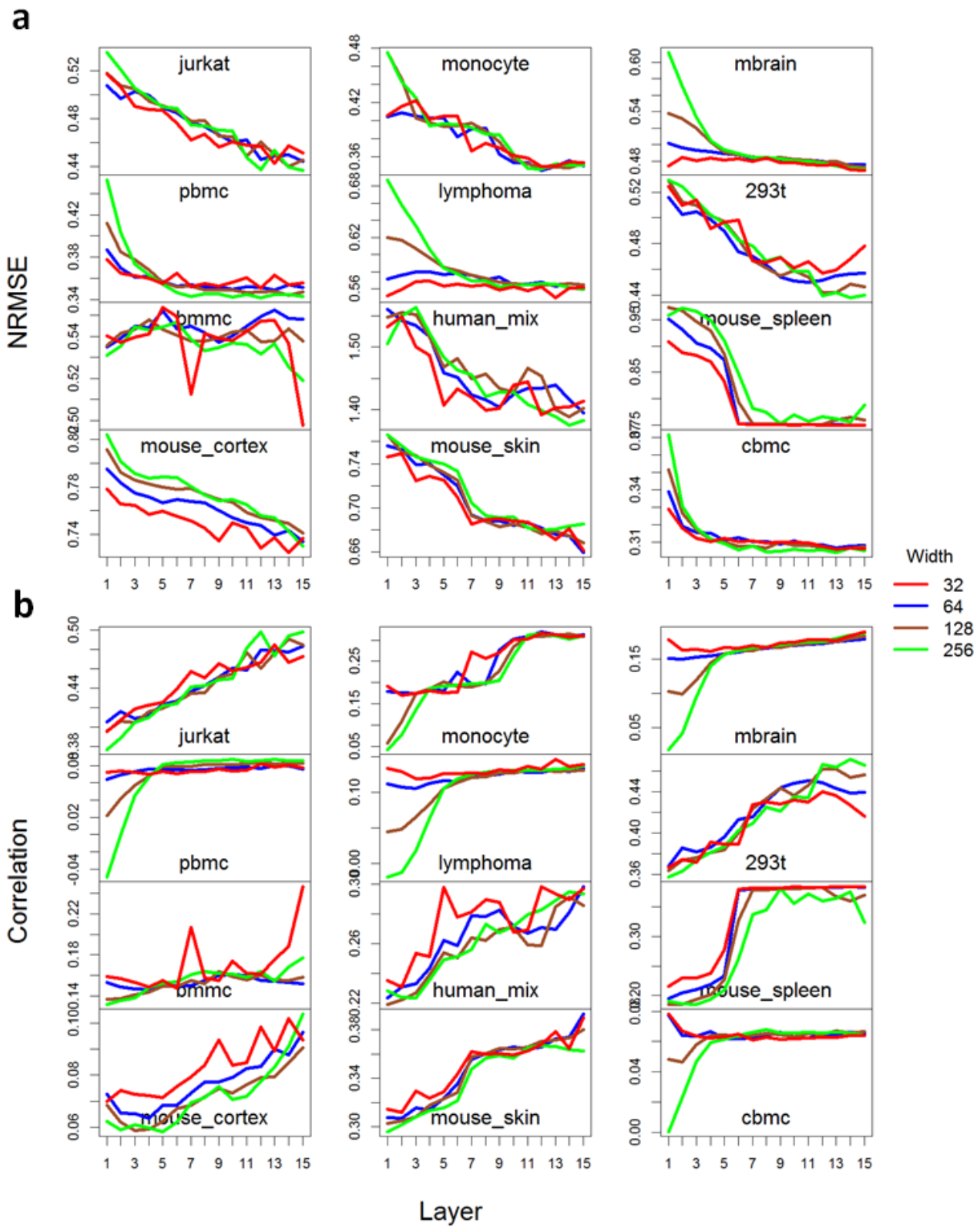
Supplementary Table S2. The 20 scRNA-seq datasets with cell type labels used to evaluate the cell clustering.

Dataset	Tissue/Cell type	Experimental protocol	# of cells	# of genes	# of cell types	Zero rate	Reference
Zhengmix4uneq	Peripheral blood mononuclear cells	10x Genomics	6498	16443	4	96.81%	[52]
Zhengmix4eq	Peripheral blood mononuclear cells	10x Genomics	3994	15568	4	96.62%	[52]
mouse_cortex	Mouse cortex and hippocampus	Fluidigm C1	3005	19972	9	81.21%	[131,132]
mouse1_umifm_counts	Mouse pancreatic islets	inDrop	822	14878	13	90.48%	[135]
mouse2_umifm_counts	Mouse pancreatic islets	inDrop	1064	14878	13	87.80%	[135]
human1_umifm_counts	Human pancreatic islets	inDrop	1937	16381	14	90.41%	[135]
human2_umifm_counts	Human pancreatic islets	inDrop	1724	20125	14	90.59%	[135]
human3_umifm_counts	Human pancreatic islets	inDrop	3605	20125	14	91.30%	[135]
human4_umifm_counts	Human pancreatic islets	inDrop	1303	16381	14	89.01%	[135]
Sliver	Peripheral blood mononuclear cells	10x Genomics	2590	58302	11	98.42%	[136]
Lake	Human brain cells	Fluidigm C1	3042	25051	16	53.69%	[137]
Li	Human colorectal tumors	SMARTer	561	55186	9	78.52%	[138]
liver	Human liver	SMARTer	777	19020	7	68.14%	[139]

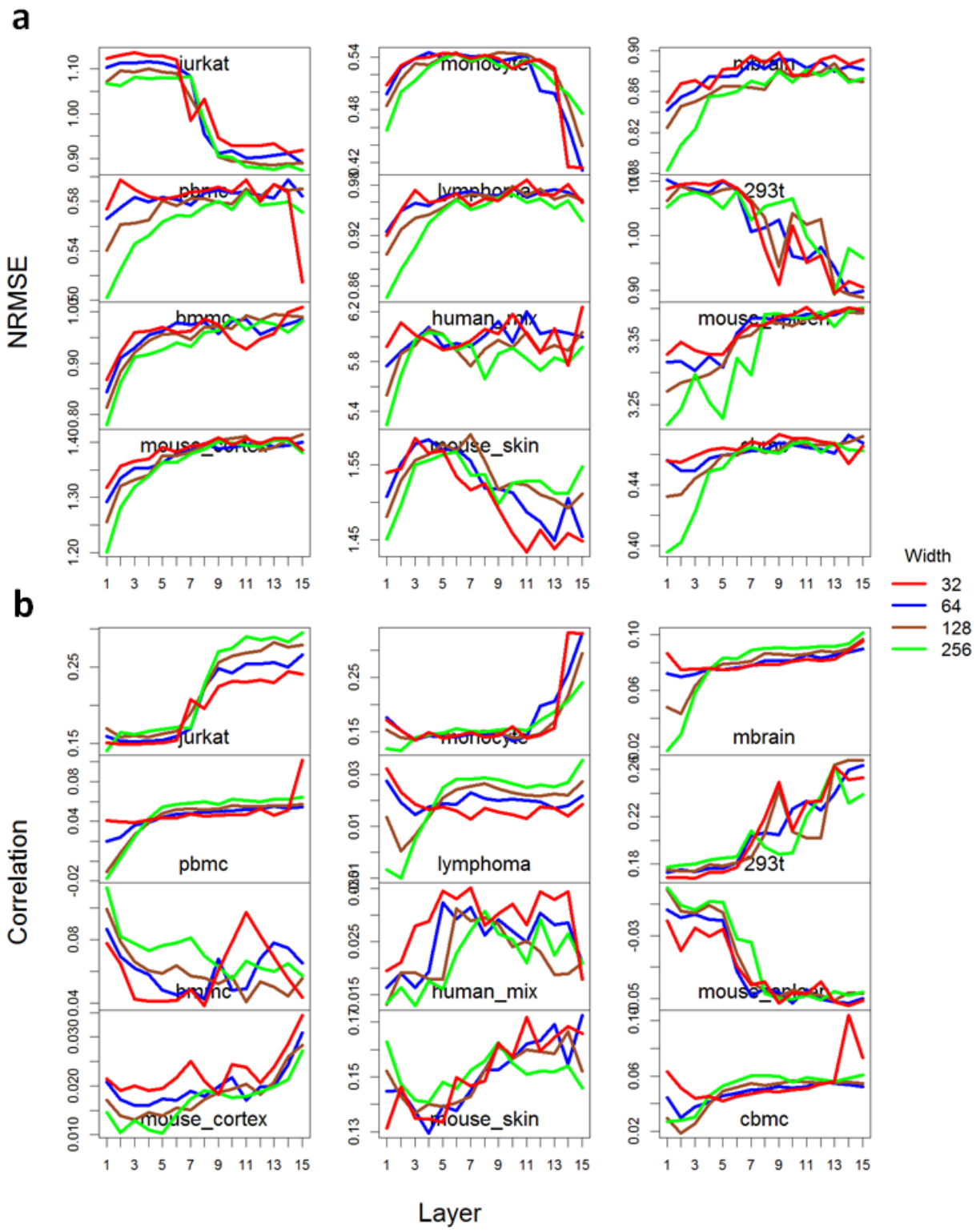
Romanov	Mouse brain cells	Drop-seq	2881	24341	7	87.72%	[140]
Camp	Human brain cells	SMARTer	734	18927	6	80.11%	[141]
manno_human	Human brain cells	STRT-Seq UMI	4029	20560	56	39.69%	[142]
Klein	mouse embryonic stem cells	inDrop	2717	24175	4	82.86%	[143]
Usoskin	Mouse brain cells	STRT-Seq	622	25334	4	39.47%	[144]
Tasic	Mouse visual cortex cells	SMARTer	1679	24150	18	68.30%	[145]
Chen	Mouse hypothalamus	Drop-seq	14437	23530	47	93.35%	[146]

Supplementary Table S3. The 20 scRNA-seq datasets that generate synthetic datasets in the evaluation of DE gene analysis.

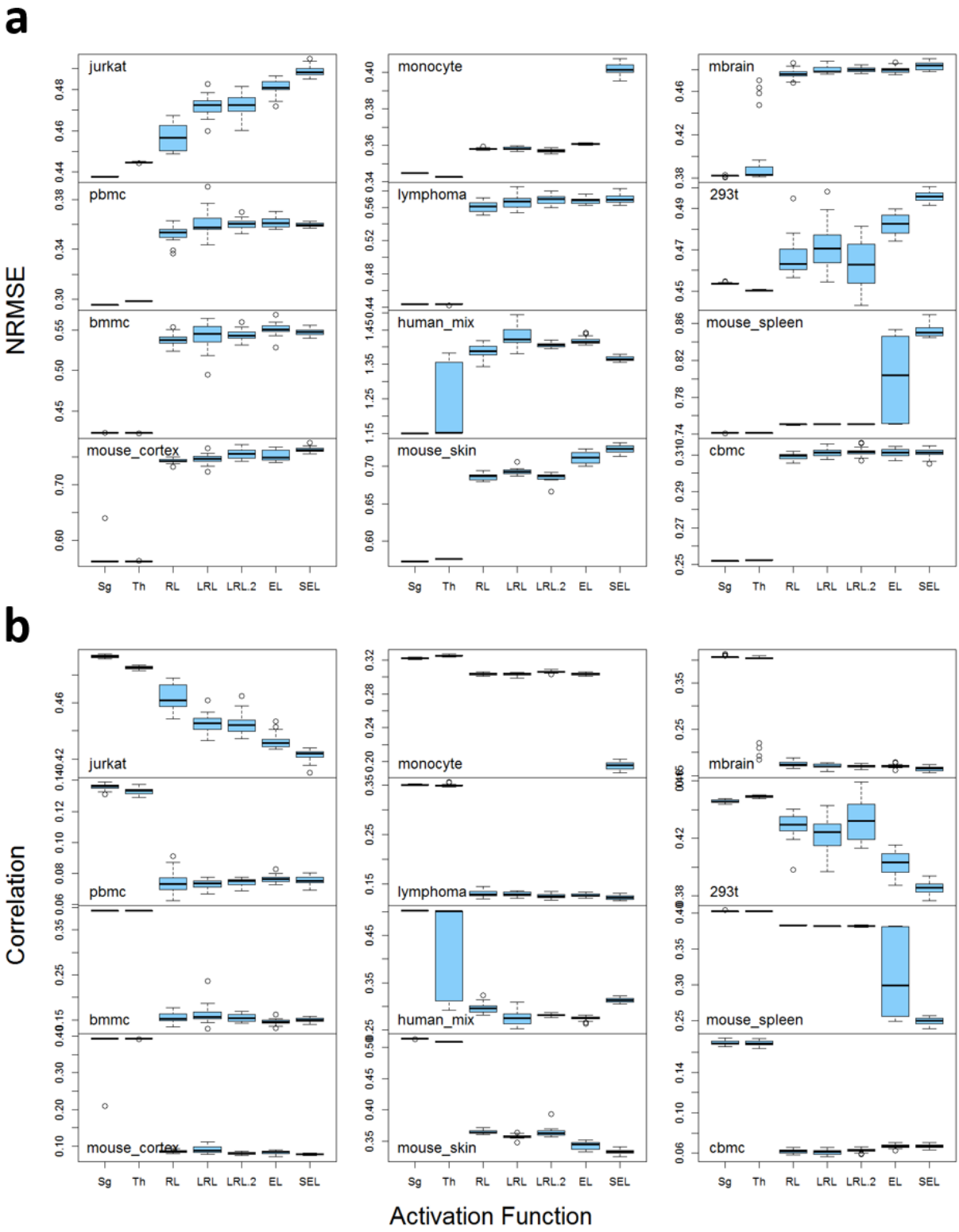
Dataset	Tissue/Cell type	Experimental protocol	# of cells	# of genes	Zero rate	Reference
T	Pan T cells	10x Genomics	3551	33694	96.60%	[52]
cd8	CD8+ Cytotoxic T cells	10x Genomics	10209	32738	98.21%	[52]
293t	293T cells	10x Genomics	2885	32738	89.56%	[52]
Acinar_cells	Acinar cells	10x Genomics	1514	30036	99.74%	[14]
Astrocytes	Astrocytes	Drop-seq	655	29651	95.42%	[14]
Oligodendrocytes	Oligodendrocytes	Drop-seq	428	29651	94.79%	[14]
Keratinocytes	Keratinocytes	10x Genomics	3241	33293	96.07%	[14]
Myoepithelial_cells	Myoepithelial cells	Drop-seq	896	28660	96.23%	[14]
Epithelial_cells	Epithelial cells	Drop-seq	611	28660	94.42%	[14]
NK	Natural killer cells	10x Genomics	2889	33321	94.52%	[14]
Endothelial_cells	Endothelial cells	10x Genomics	861	32925	94.04%	[14]
Hematopoietic_stem_cells	Hematopoietic stem cells	SMART-seq2	1386	38240	88.08%	[14]
Fibroblasts	Fibroblasts	SMART-seq2	2238	47873	92.94%	[14]
Macrophages	Macrophages	SMART-seq2	744	47873	93.42%	[14]
Interneurons	Interneurons	inDrops	2607	35443	96.11%	[14]
Foveolar_cells	Foveolar cells	Microwell-seq	503	18521	95.49%	[14]
Neutrophils	Neutrophils	Microwell-seq	1049	19460	96.95%	[14]
Neurons	Mouse neurons	Drop-seq	641	28142	92.72%	[14]
Tanycytes	Tanycytes	Drop-seq	507	28142	95.52%	[14]
Pulmonary	Pulmonary alveolar type I cells	10x Genomics	727	27140	91.27%	[14]



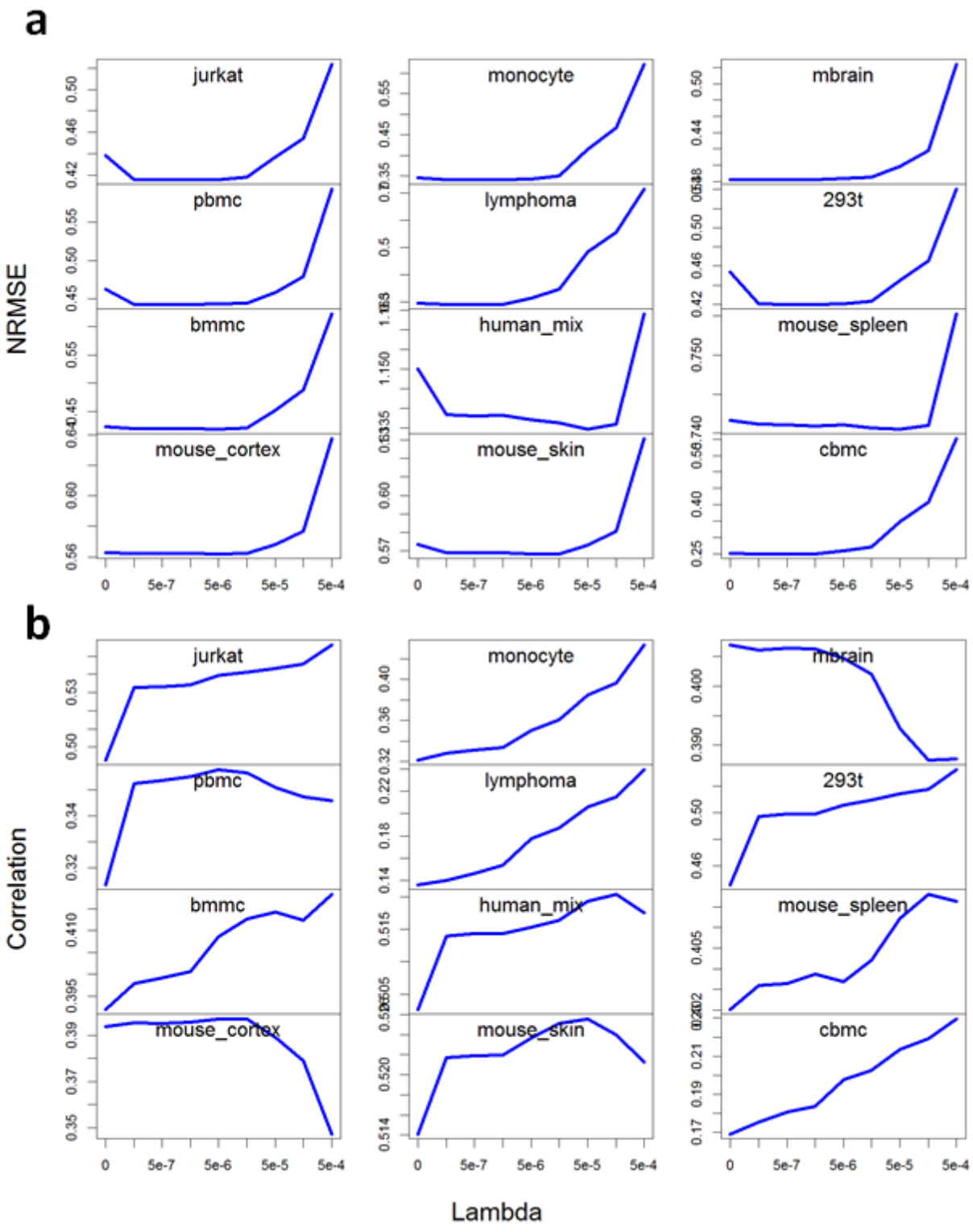
Supplementary figure S1. The impact of depth and width on the imputation NRMSE (**a**) and correlation (**b**) based on the double exponential masking scheme.



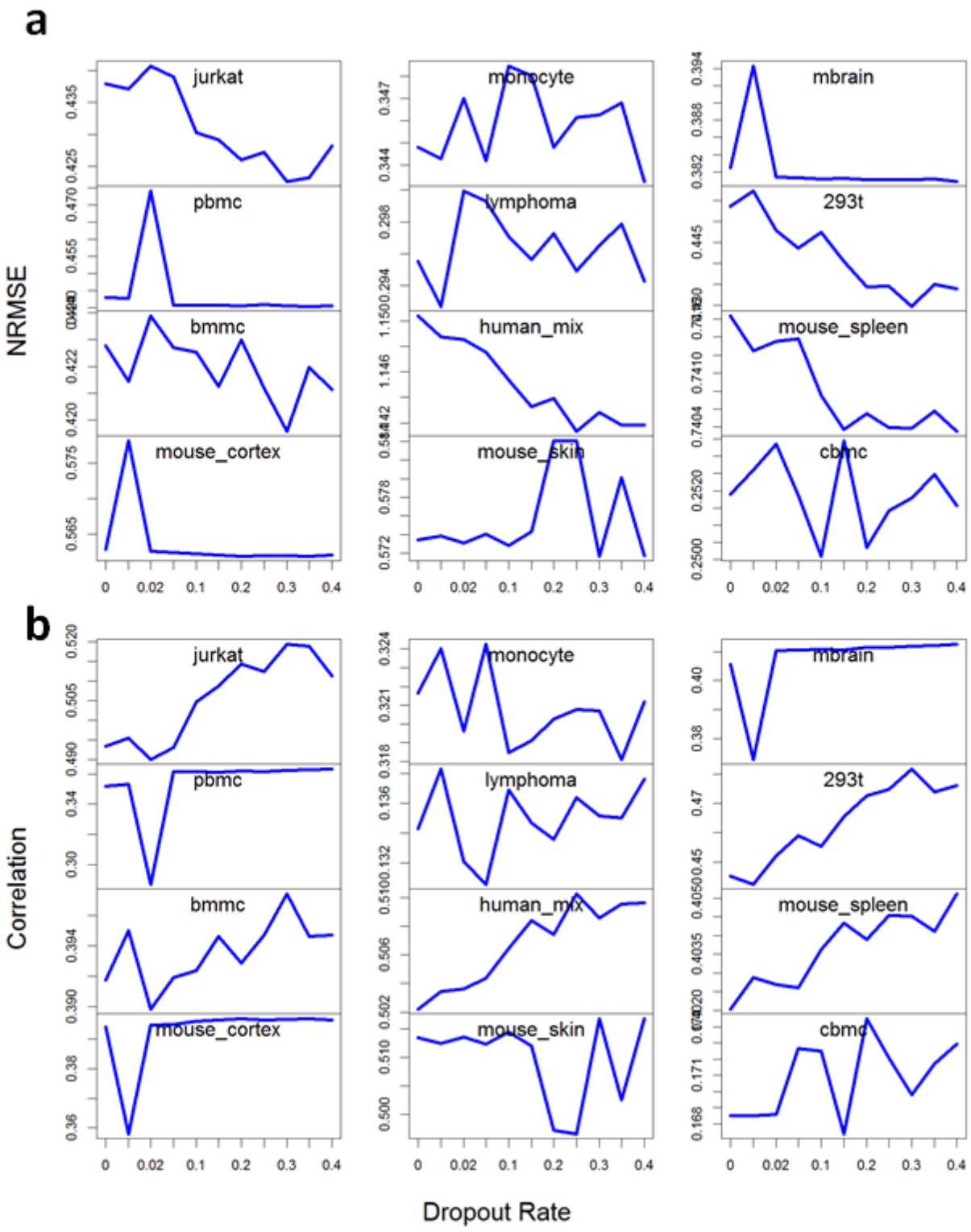
Supplementary figure S2. The impact of depth and width on the imputation NRMSE (**a**) and correlation (**b**) based on the median masking scheme.



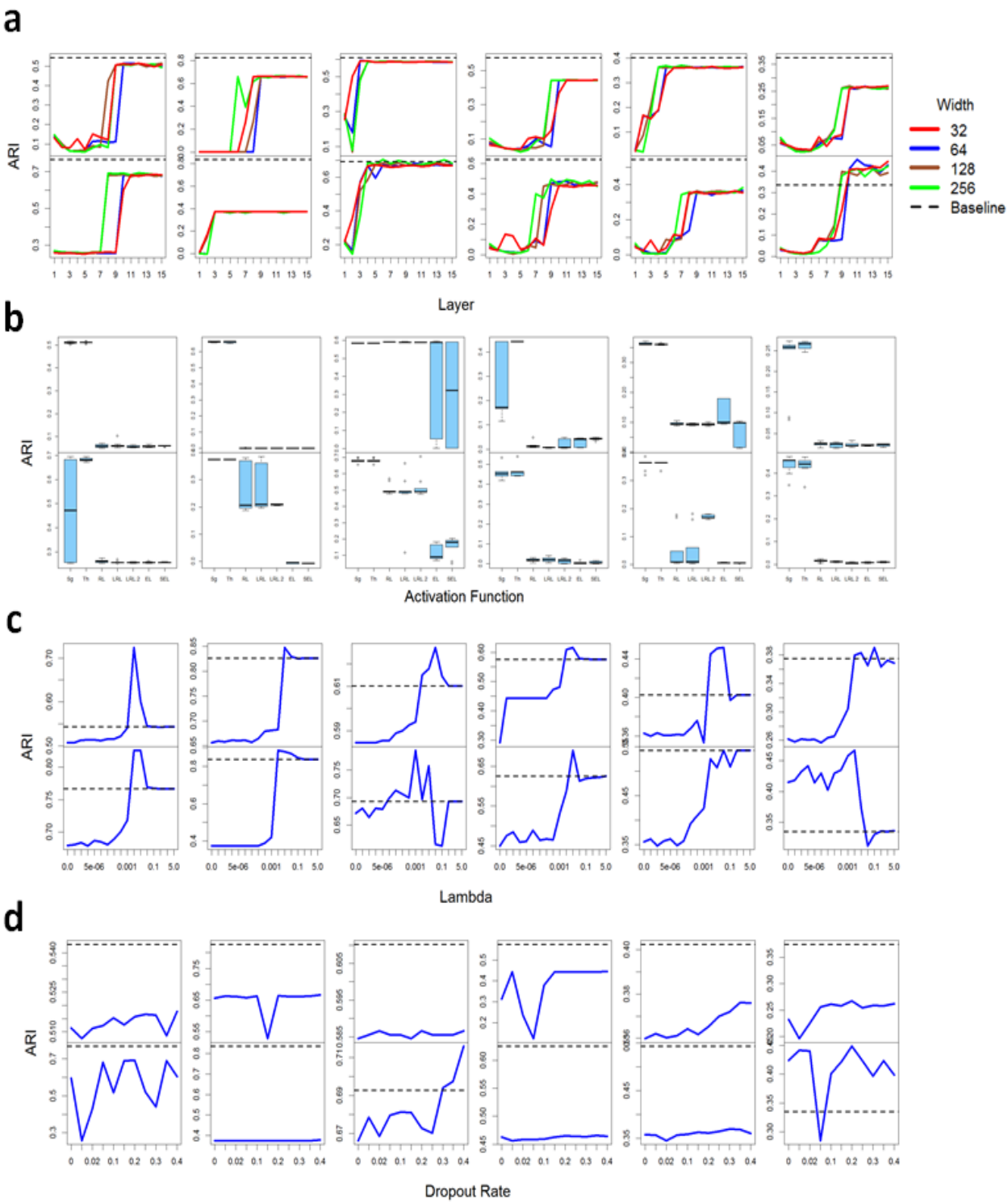
Supplementary Figure S3. The impact of activation functions on the imputation NRMSE (**a**) and correlation (**b**) based on the double exponential masking scheme. Sg: sigmoid; Th: tanh; RL: ReLU; LRL: LeakyReLU ($\alpha = 0.01$); LRL: LeakyReLU.2 ($\alpha = 0.2$); EL: ELU; SEL: SELU.



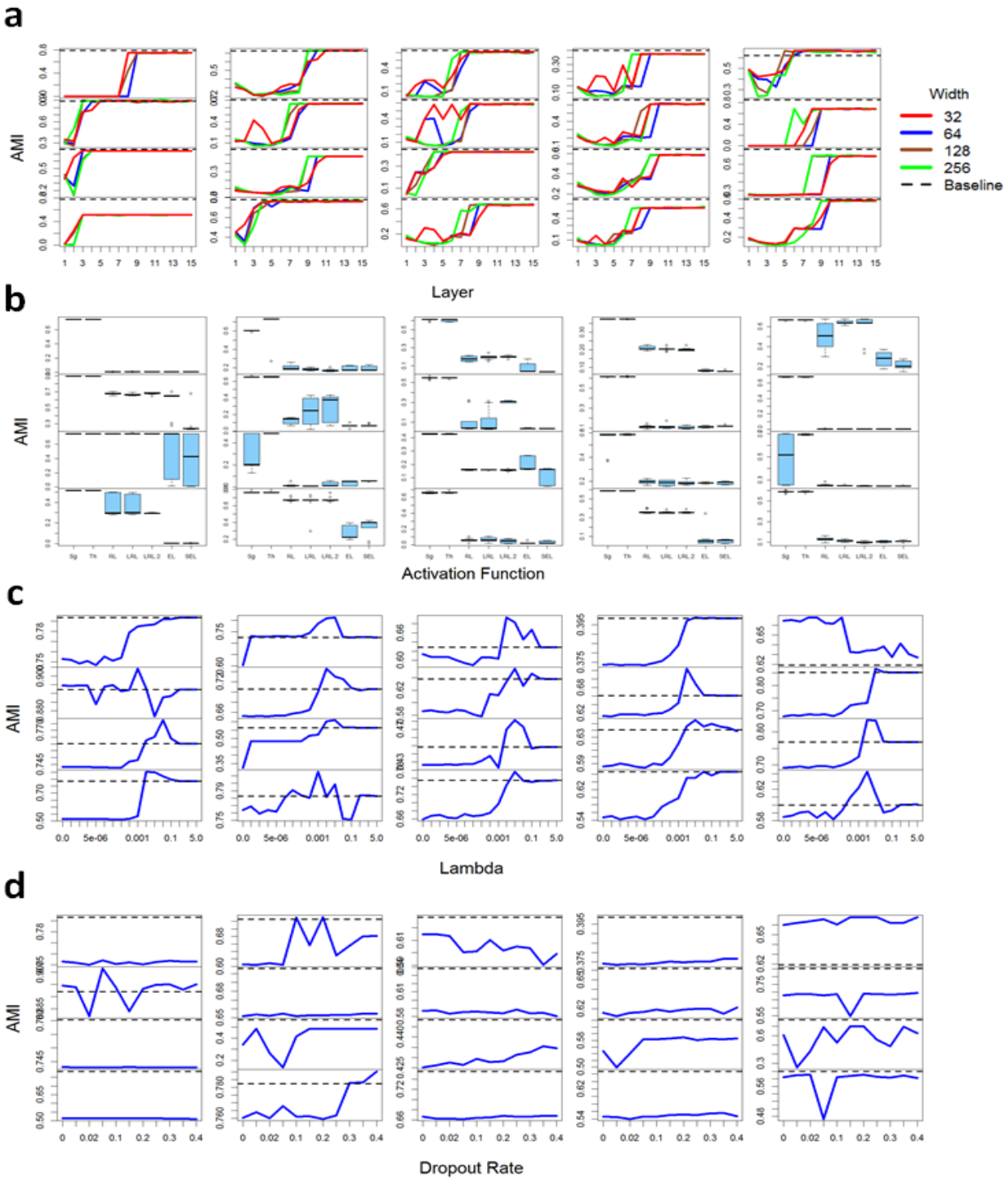
Supplementary Figure S4. The impact of weight decay regularization on the imputation NRMSE (**a**) and correlation (**b**) based on the double exponential masking scheme.



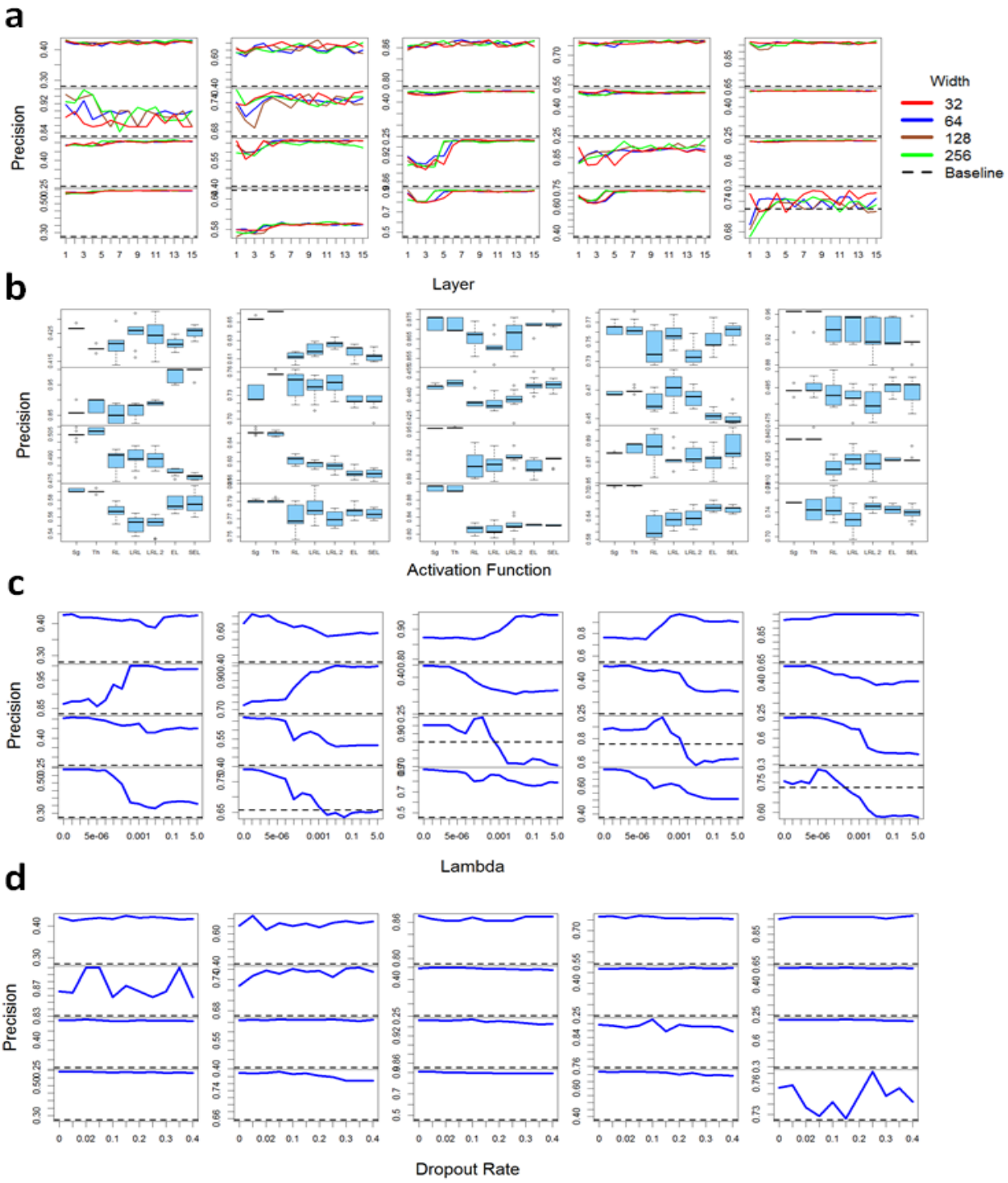
Supplementary Figure S5. The impact of dropout regularization on the imputation NRMSE (**a**) and correlation (**b**) based on the double exponential masking scheme.



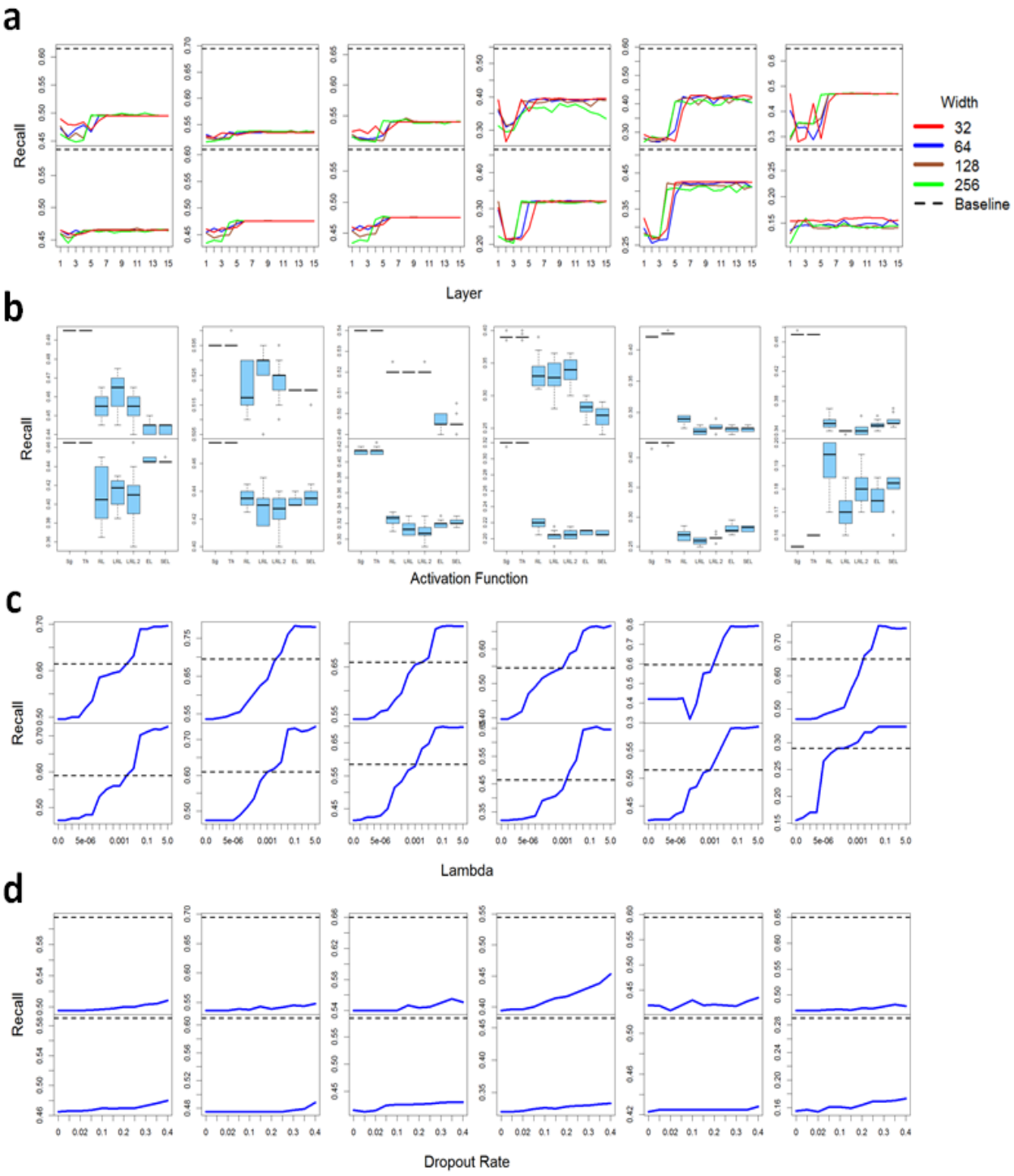
Supplementary Figure S6. The impact of autoencoder design on the cell clustering measured by ARI (continued). **a,** Depth and width. **b**,Activation function. **c**, Weight decay regularization. **d**, Dropout regularization. 12 datasets are shown here (from left to right; from top to bottom): human3_umifm_counts, Zhengmix4eq, liver, romanov, camp, manno_human, klein, usoskin, tasic, human1_umifm_counts, mouse2_umifm_counts, chen.



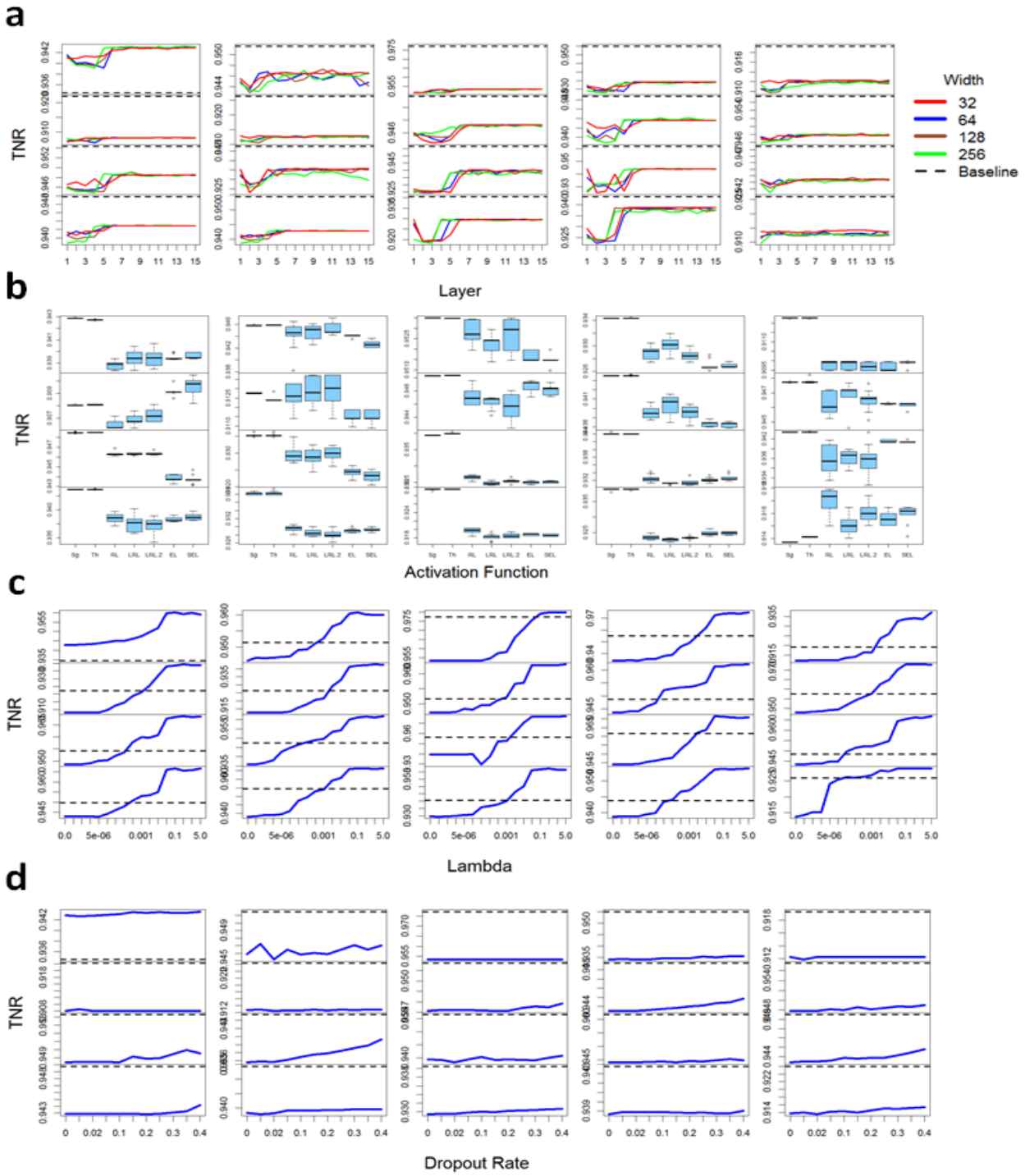
Supplementary Figure S7. The impact of autoencoder design on the cell clustering measured by AMI (continued). **a,** Depth and width. **b**, Activation function. **c**, Weight decay regularization. **d**, Dropout regularization. 20 datasets are shown here (from left to right; from top to bottom): Zhengmix4uneq, Zeisel, mouse1_umifm_counts, Silver, lake, li, human2_umifm_counts, human4_umifm_counts, human3_umifm_counts, Zhengmix4eq, liver, romanov, camp, manno_human, klein, usoskin, tasic, human1_umifm_counts, mouse2_umifm_counts, chen.



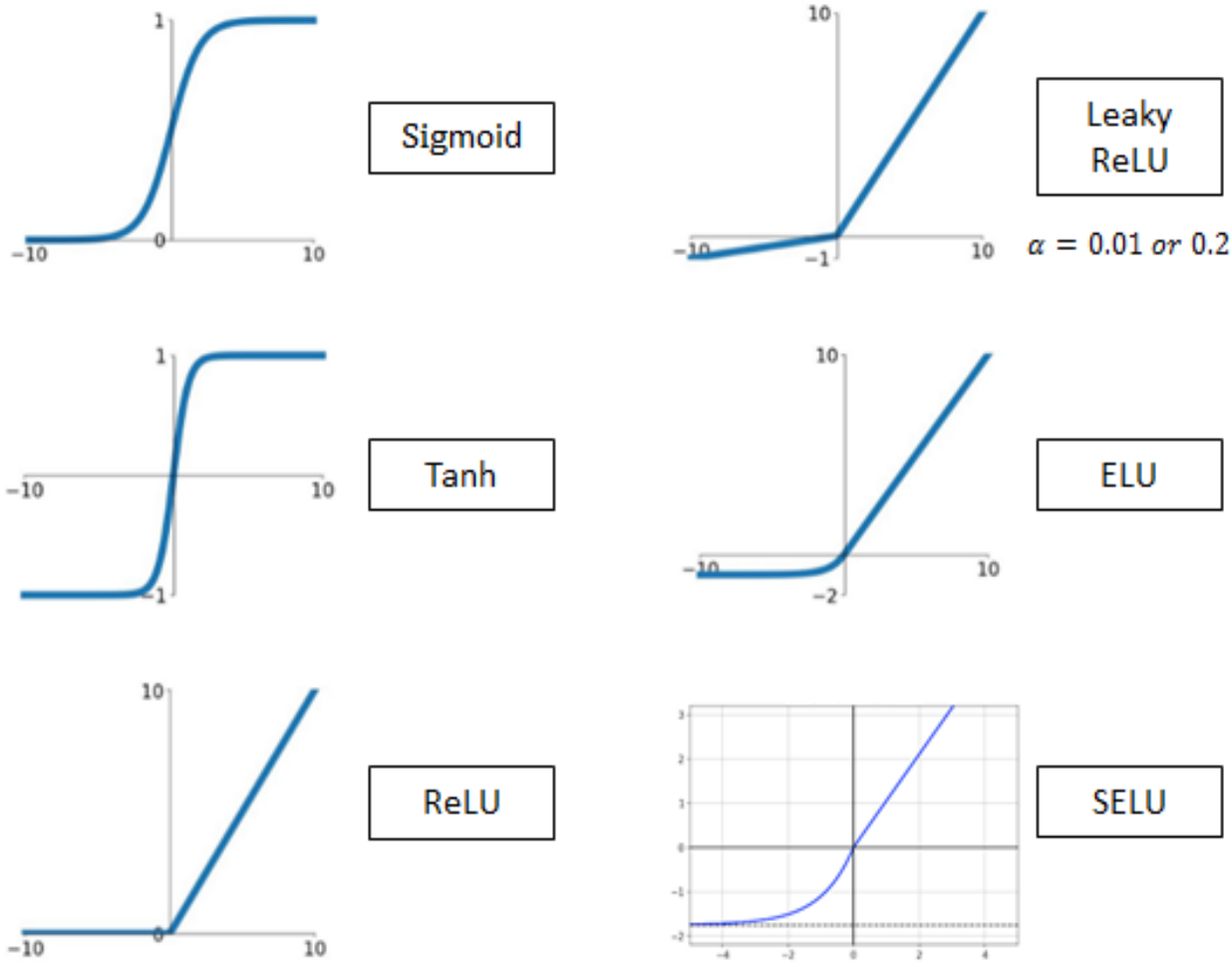
Supplementary Figure S8. The impact of autoencoder design on the DE gene analysis measured by precision. **a,** Depth and width. **b**,Activation function. **c**, Weight decay regularization. **d**, Dropout regularization. 20 datasets are shown here (from left to right; from top to bottom): T, cd8, 293t, Fibroblasts, Macrophages, Endothelial_cells, Hematopoietic_stem_cells, NK, Keratinocytes, Neurons, Pulmonary, Myoepithelial_cells, Interneurons, Oligodendrocytes, Neutrophils, Foveolar_cells, Epithelial_cells, Tanycytes, Astrocytes, Acinar_cells .



Supplementary Figure S9. The impact of autoencoder design on the DE gene analysis measured by recall (continued). **a,** Depth and width. **b**,Activation function. **c**, Weight decay regularization. **d**, Dropout regularization. 12 datasets are shown here (from left to right; from top to bottom): Keratinocytes, Neurons, Pulmonary, Myoepithelial_cells, Interneurons, Oligodendrocytes, Neutrophils, Foveolar_cells, Epithelial_cells, Tanycytes, Astrocytes, Acinar_cells.



Supplementary Figure S10. The impact of autoencoder design on the DE gene analysis measured by TNR. **a,** Depth and width. **b**,Activation function. **c**, Weight decay regularization. **d**, Dropout regularization. 20 datasets are shown here (from left to right; from top to bottom): T, cd8, 293t, Fibroblasts, Macrophages, Endothelial_cells, Hematopoietic_stem_cells, NK, Keratinocytes, Neurons, Pulmonary, Myoepithelial_cells, Interneurons, Oligodendrocytes, Neutrophils, Foveolar_cells, Epithelial_cells, Tanycytes, Astrocytes, Acinar_cells.



Supplementary Figure S11. The seven activation functions evaluated in this study. The LeakyReLU activation function has two hyperparameter settings: $\alpha = 0.01$and $\alpha = 0.2$.

CHAPTER 5

Conclusions

Single-cell RNA sequencing (scRNA-seq) has brought up enormous opportunities and challenges. The scRNA-seq community starts to name the scRNA-seq data analysis as single-cell data science (SCDS) [147]. SCDS tries to handle those challenges in a statistically solid and computationally efficient fashion. To echo this trend, we attempt to answer three critical questions in this dissertation: how to systematically benchmark the computational doublet detection methods; how to automate such benchmark so that it can accommodate the fast-growing methods efficiently; how to design autoencoder-based imputation method to denoise scRNA-seq data. Our results provide promising solutions to those questions.

There are several issues in scRNA-seq data analysis which have not been discussed thoroughly. We plan to explore potential solutions in future work. First, how to construct/infer/predict the whole-genome gene expression of single cells under different time points or experimental conditions. In principle, scRNA-seq data is a snapshot of single cells' genomic appearance. The measurement of gene expression in one cell requires dissolving that cell, which means it is infeasible to remeasure its gene expression

thereafter. This cell lysis process excludes the possibility to obtain the single cells' time-series gene expression or their gene expression under both treatment and control. With those data, however, researchers can construct the real cell developmental trajectory and the causal effect of treatment. Although machine learning methods can be used to construct/infer/predict those "counterfactual" data [148], the major challenge is how to validate the result due to the lack of ground truth. Fluorescence in situ hybridization (FISH)-based experimental protocols may provide expression measurements for a small number of genes across time or conditions [149]. Yet how to validate the machine learning models on the rest of the genes remains a challenge.

Second, how to design a subsampling method that can improve the computational efficiency and downstream analysis simultaneously. A scRNA-seq dataset may contain up to millions of cells, each of which has expression levels up to thousands of genes. Analyzing such huge datasets is often beyond the capacity of a single computer. A subsample is necessary to provide a computable subset with the essential information preserved from the full data. The naive random subsampling has limited impacts on downstream analysis, especially cell clustering since it does not change the distribution of cell types in the full data. A better subsampling method should improve cell clustering by generating samples with more balanced cell types than the full data. Moreover, a balanced subsample will benefit the identification of rare cell types since it includes more of them than the full data and random sample. A potential solution is to use space-filling design [150–155,159] to select cells close to the center of each cluster without the information of cell types. The subsample will mainly cover cells close to the center of each cluster and thus improve the performance of the clustering algorithm.

Finally, how to effectively utilize innovative hardware, especially the graphics processing unit (GPU), to improve the computational efficiency of current and future methods. The increasingly large number of cells and genes in scRNA-seq data has posed a great computational challenge on current methods that are mainly developed in the R programming language. In scientific computing, GPUs are used to accelerate the training of deep neural networks in computer vision and natural language processing. Although some deep learning methods have been developed for scRNA-seq data, they are limited to certain machine learning tasks, for example, imputation and dimension reduction [24]. The field lacks a comprehensive software package with GPU acceleration that includes the life-cycle functionalities of scRNA-seq data analysis. The recently-developed general-purpose Python libraries with GPU support (e.g., Pytorch [125] and CuPy [156]) make it feasible to reimplement popular methods on GPUs. Such reimplementation will significantly increase the computational efficiency of current methods and handle the massive scRNA-seq data in the future.

References

1. Saliba, A.-E., Westermann, A. J., Gorski, S. A. & Vogel, J. Single-cell RNA-seq: advances and future challenges. *Nucleic Acids Research* vol. 42 8845–8860 (2014).
2. Vallejos, C. A., Marioni, J. C. & Richardson, S. BASiCS: Bayesian Analysis of Single-Cell Sequencing Data. *PLoS Comput. Biol.* **11**, e1004333 (2015).
3. Kolodziejczyk, A. A., Kim, J. K., Svensson, V., Marioni, J. C. & Teichmann, S. A. The technology and biology of single-cell RNA sequencing. *Mol. Cell* **58**, 610–620 (2015).
4. Liu, S. & Trapnell, C. Single-cell transcriptome sequencing: recent advances and remaining challenges. *F1000Res.* **5**, (2016).
5. Luecken, M. D. & Theis, F. J. Current best practices in single-cell RNA-seq analysis: a tutorial. *Mol. Syst. Biol.* **15**, (2019).
6. Han, X. *et al.* Mapping the Mouse Cell Atlas by Microwell-Seq. *Cell* **173**, 1307 (2018).
7. Cao, J. *et al.* The single-cell transcriptional landscape of mammalian organogenesis. *Nature* **566**, 496–502 (2019).
8. Hwang, B., Lee, J. H. & Bang, D. Single-cell RNA sequencing technologies and bioinformatics pipelines. *Exp. Mol. Med.* **50**, 96 (2018).
9. Chen, G., Ning, B. & Shi, T. Single-Cell RNA-Seq Technologies and Related Computational Data Analysis. *Front. Genet.* **10**, 317 (2019).
10. Picelli, S. *et al.* Smart-seq2 for sensitive full-length transcriptome profiling in single

cells. *Nat. Methods* **10**, 1096–1098 (2013).

11. Svensson, V. *et al.* Power analysis of single-cell RNA-sequencing experiments. *Nat. Methods* **14**, 381–387 (2017).
12. Ziegenhain, C. *et al.* Comparative Analysis of Single-Cell RNA Sequencing Methods. *Mol. Cell* **65**, 631–643.e4 (2017).
13. Regev, A. *et al.* Science forum: the human cell atlas. *Elife* **6**, e27041 (2017).
14. Franzén, O., Gan, L.-M. & Björkegren, J. L. M. PanglaoDB: a web server for exploration of mouse and human single-cell RNA sequencing data. *Database* **2019**, (2019).
15. Andrews, T. S., Kiselev, V. Y., McCarthy, D. & Hemberg, M. Tutorial: guidelines for the computational analysis of single-cell RNA sequencing data. *Nat. Protoc.* **16**, 1–9 (2021).
16. Zappia, L., Phipson, B. & Oshlack, A. Exploring the single-cell RNA-seq analysis landscape with the scRNA-tools database. *PLoS Comput. Biol.* **14**, e1006245 (2018).
17. Pierre-Luc. *scDblFinder*. (GitHub).
18. Tian, L. *et al.* Benchmarking single cell RNA-sequencing analysis pipelines using mixture control experiments. *Nat. Methods* **16**, 479–487 (2019).
19. Duò, A., Robinson, M. D. & Soneson, C. A systematic performance evaluation of clustering methods for single-cell RNA-seq data. *F1000Res.* **7**, 1141 (2018).
20. Saelens, W., Cannoodt, R., Todorov, H. & Saeys, Y. A comparison of single-cell trajectory inference methods: towards more accurate and robust tools. doi:10.1101/276907.

21. Tran, H. T. N. *et al.* A benchmark of batch-effect correction methods for single-cell RNA sequencing data. *Genome Biol.* **21**, 12 (2020).
22. Hou, W., Ji, Z., Ji, H. & Hicks, S. C. A systematic evaluation of single-cell RNA-sequencing imputation methods. *Genome Biol.* **21**, 218 (2020).
23. Abdelaal, T. *et al.* A comparison of automatic cell identification methods for single-cell RNA sequencing data. *Genome Biol.* **20**, 194 (2019).
24. Lähnemann, D. *et al.* Eleven grand challenges in single-cell data science. *Genome Biol.* **21**, 31 (2020).
25. Miles Xi, N. & Li, J. J. Protocol for Benchmarking Computational Doublet-Detection Methods in Single-Cell RNA Sequencing Data Analysis. *arXiv e-prints* arXiv:2101.08860 (2021).
26. Jiang, R., Sun, T., Song, D. & Li, J. J. Zeros in scRNA-seq data: good or bad? How to embrace or tackle zeros in scRNA-seq data analysis? *bioRxiv* (2020).
27. Azizi, E., Prabhakaran, S., Carr, A. & Pe'er, D. Bayesian Inference for Single-cell Clustering and Imputing. *Genomics and Computational Biology* vol. 3 46 (2017).
28. Li, W. V. & Li, J. J. An accurate and robust imputation method scImpute for single-cell RNA-seq data. *Nat. Commun.* **9**, 997 (2018).
29. van Dijk, D. *et al.* Recovering Gene Interactions from Single-Cell Data Using Data Diffusion. *Cell* vol. 174 716–729.e27 (2018).
30. Wagner, F., Yan, Y. & Yanai, I. K-nearest neighbor smoothing for high-throughput single-cell RNA-Seq data. *BioRxiv* (2017).
31. Lopez, R., Regier, J., Cole, M. B., Jordan, M. I. & Yosef, N. Deep generative modeling for single-cell transcriptomics. *Nat. Methods* **15**, 1053–1058 (2018).

32. Eraslan, G., Simon, L. M., Mircea, M., Mueller, N. S. & Theis, F. J. Single-cell RNA-seq denoising using a deep count autoencoder. *Nat. Commun.* **10**, 390 (2019).
33. Zhang, L. & Zhang, S. Comparison of computational methods for imputing single-cell RNA-sequencing data. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **17**, 376–389 (2020).
34. Xi, N. *et al.* Understanding the Political Ideology of Legislators from Social Media Images. *ICWSM* **14**, 726–737 (2020).
35. Nair, V. & Hinton, G. E. Rectified linear units improve restricted boltzmann machines. in *Icml* (2010).
36. Nwankpa, C., Ijomah, W., Gachagan, A. & Marshall, S. Activation Functions: Comparison of trends in Practice and Research for Deep Learning. *arXiv [cs.LG]* (2018).
37. Goodfellow, I., Bengio, Y., Courville, A. & Bengio, Y. *Deep learning*. vol. 1 (MIT press Cambridge, 2016).
38. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **15**, 1929–1958 (2014).
39. Bernstein, N. J. *et al.* Solo: Doublet Identification in Single-Cell RNA-Seq via Semi-Supervised Deep Learning. *Cell Syst* **11**, 95–101.e5 (2020).
40. Wolock, S. L., Lopez, R. & Klein, A. M. Scrublet: Computational Identification of Cell Doublets in Single-Cell Transcriptomic Data. *Cell Syst* **8**, 281–291.e9 (2019).
41. McGinnis, C. S., Murrow, L. M. & Gartner, Z. J. DoubletFinder: Doublet Detection in Single-Cell RNA Sequencing Data Using Artificial Nearest Neighbors. *Cell Syst* **8**,

329–337.e4 (2019).

42. Stoeckius, M. *et al.* Cell Hashing with barcoded antibodies enables multiplexing and doublet detection for single cell genomics. *Genome Biol.* **19**, 224 (2018).
43. Kang, H. M. *et al.* Multiplexed droplet single-cell RNA-sequencing using natural genetic variation. *Nat. Biotechnol.* **36**, 89–94 (2018).
44. McGinnis, C. S. *et al.* MULTI-seq: sample multiplexing for single-cell RNA sequencing using lipid-tagged indices. *Nature Methods* vol. 16 619–626 (2019).
45. Bais, A. S. & Kostka, D. scds: computational annotation of doublets in single-cell RNA sequencing data. *Bioinformatics* (2019) doi:10.1093/bioinformatics/btz698.
46. DePasquale, E. A. K. *et al.* DoubletDecon: Deconvoluting Doublets from Single-Cell RNA-Sequencing Data. *Cell Rep.* **29**, 1718–1727.e8 (2019).
47. Gayoso, A. & Shor, J. DoubletDetection. *Zenodo* (2018).
48. Lun, A. T. L., McCarthy, D. J. & Marioni, J. C. A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor. *F1000Research* vol. 5 2122 (2016).
49. Branco, P., Torgo, L. & Ribeiro, R. P. A Survey of Predictive Modeling on Imbalanced Domains. (2016).
50. Bloom, J. D. Estimating the frequency of multiplets in single-cell RNA sequencing from cell-mixing experiments. *PeerJ* **6**, e5578 (2018).
51. Li, W. V. & Li, J. J. A statistical simulator scDesign for rational scRNA-seq experimental design. *Bioinformatics* **35**, i41–i50 (2019).
52. Zheng, G. X. Y. *et al.* Massively parallel digital transcriptional profiling of single cells. *Nat. Commun.* **8**, 14049 (2017).

53. Saito, T. & Rehmsmeier, M. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLoS One* **10**, e0118432 (2015).
54. https://github.com/EDePasquale/DoubletDecon/issues.
55. Love, M. I., Huber, W. & Anders, S. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biol.* **15**, 550 (2014).
56. Finak, G. *et al.* MAST: a flexible statistical framework for assessing transcriptional changes and characterizing heterogeneity in single-cell RNA sequencing data. *Genome Biol.* **16**, 278 (2015).
57. Fay, M. P. & Proschan, M. A. Wilcoxon-Mann-Whitney or t-test? On assumptions for hypothesis tests and multiple interpretations of decision rules. *Stat. Surv.* **4**, 1–39 (2010).
58. Wang, T., Li, B., Nelson, C. E. & Nabavi, S. Comparative analysis of differential gene expression analysis tools for single-cell RNA sequencing data. *BMC Bioinformatics* **20**, 40 (2019).
59. Yip, S. H., Sham, P. C. & Wang, J. Evaluation of tools for highly variable gene discovery from single-cell RNA-seq data. *Brief. Bioinform.* **20**, 1583–1589 (2019).
60. Amezquita, R. A. *et al.* Orchestrating single-cell analysis with Bioconductor. *Nature Methods* (2019) doi:10.1038/s41592-019-0654-x.
61. Butler, A., Hoffman, P., Smibert, P., Papalexi, E. & Satija, R. Integrating single-cell transcriptomic data across different conditions, technologies, and species. *Nat. Biotechnol.* **36**, 411–420 (2018).
62. Stuart, T. *et al.* Comprehensive Integration of Single-Cell Data. *Cell* **177**, 1888–

1902.e21 (2019).

63. Blondel, V. D., Guillaume, J.-L., Lambiotte, R. & Lefebvre, E. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* vol. 2008 P10008 (2008).
64. Ester, M., Kriegel, H.-P., Sander, J., Xu, X. & Others. A density-based algorithm for discovering clusters in large spatial databases with noise. in *Kdd* vol. 96 226–231 (1996).
65. Feng, C. *et al.* Dimension Reduction and Clustering Models for Single-Cell RNA Sequencing Data: A Comparative Study. *Int. J. Mol. Sci.* **21**, (2020).
66. Trapnell, C. *et al.* The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells. *Nat. Biotechnol.* **32**, 381–386 (2014).
67. Zappia, L., Phipson, B. & Oshlack, A. Splatter: simulation of single-cell RNA sequencing data. *Genome Biol.* **18**, 174 (2017).
68. Street, K. *et al.* Slingshot: cell lineage and pseudotime inference for single-cell transcriptomics. *BMC Genomics* vol. 19 (2018).
69. Herring, C. A., Chen, B., McKinley, E. T. & Lau, K. S. Single-Cell Computational Strategies for Lineage Reconstruction in Tissue Systems. *Cell Mol Gastroenterol Hepatol* **5**, 539–548 (2018).
70. Hastie, T. J. & Tibshirani, R. J. *Generalized Additive Models*. (CRC Press, 1990).
71. Ji, Z. & Ji, H. TSCAN: Pseudo-time reconstruction and evaluation in single-cell RNA-seq analysis. *Nucleic Acids Res.* **44**, e117 (2016).
72. Mangul, S., Martin, L. S., Eskin, E. & Blekhman, R. Improving the usability and archival stability of bioinformatics software. *Genome Biol.* **20**, 47 (2019).

73. Risso, D., Perraudeau, F., Gribkova, S., Dudoit, S. & Vert, J.-P. A general and flexible method for signal extraction from single-cell RNA-seq data. *Nat. Commun.* **9**, 284 (2018).
74. Weber, L. M. *et al.* Essential guidelines for computational method benchmarking. *Genome Biol.* **20**, 125 (2019).
75. Andrews, T. S. & Hemberg, M. False signals induced by single-cell imputation. *F1000Research* vol. 7 1740 (2018).
76. Efron, B. & Hastie, T. *Computer Age Statistical Inference*. (Cambridge University Press, 2016).
77. Young, M. D. & Behjati, S. SoupX removes ambient RNA contamination from droplet based single cell RNA sequencing data. *BioRxiv* (2020).
78. Yang, S. *et al.* Decontamination of ambient RNA in single-cell RNA-seq with DecontX. *Genome Biology* vol. 21 (2020).
79. Nettleton, D. F., Orriols-Puig, A. & Fornells, A. A study of the effect of different types of noise on the precision of supervised learning techniques. *Artificial Intelligence Review* **33**, 275–306 (2010).
80. Domingues, R., Filippone, M., Michiardi, P. & Zouaoui, J. A comparative evaluation of outlier detection algorithms: Experiments and analyses. *Pattern Recognit.* **74**, 406–421 (2018).
81. Natarajan, N., Dhillon, I. S., Ravikumar, P. K. & Tewari, A. Learning with Noisy Labels. in *Advances in Neural Information Processing Systems 26* (eds. Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z. & Weinberger, K. Q.) 1196–1204 (Curran Associates, Inc., 2013).

82. Dietterich, T. G. Ensemble Methods in Machine Learning. in *Multiple Classifier Systems* 1–15 (Springer Berlin Heidelberg, 2000).
83. Hastie, T., Tibshirani, R. & Friedman, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. (Springer Science & Business Media, 2009).
84. Chen, T. & Guestrin, C. XGBoost: A Scalable Tree Boosting System. in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 785–794 (Association for Computing Machinery, 2016).
85. Feurer, M. & Hutter, F. Hyperparameter Optimization. in *Automated Machine Learning: Methods, Systems, Challenges* (eds. Hutter, F., Kotthoff, L. & Vanschoren, J.) 3–33 (Springer International Publishing, 2019).
86. Waring, J., Lindvall, C. & Umeton, R. Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artif. Intell. Med.* **104**, 101822 (2020).
87. Edgar, R., Domrachev, M. & Lash, A. E. Gene Expression Omnibus: NCBI gene expression and hybridization array data repository. *Nucleic Acids Res.* **30**, 207–210 (2002).
88. Durinck, S., Spellman, P. T., Birney, E. & Huber, W. Mapping identifiers for the integration of genomic datasets with the R/Bioconductor package biomaRt. *Nat. Protoc.* **4**, 1184–1191 (2009).
89. Pfister, R., Schwarz, K. A., Janczyk, M., Dale, R. & Freeman, J. B. Good things peak in pairs: a note on the bimodality coefficient. *Front. Psychol.* **4**, 700 (2013).
90. Gong, T. & Szustakowski, J. D. DeconRNASeq: a statistical framework for

deconvolution of heterogeneous tissue samples based on mRNA-Seq data. *Bioinformatics* **29**, 1083–1085 (2013).

91. Xi, N. M. & Li, J. J. Benchmarking Computational Doublet-Detection Methods for Single-Cell RNA Sequencing Data. *Cell Systems* **12**, 176–194.e6 (2021).
92. Gayoso, A. & Shor, J. GitHub: DoubletDetection. (2019).
93. Germain, P.-L., Sonrel, A. & Robinson, M. D. pipeComp, a general framework for the evaluation of computational pipelines, reveals performant single cell RNA-seq preprocessing tools. *Genome Biol.* **21**, 227 (2020).
94. McDavid, A. *et al.* Data exploration, quality control and testing in single-cell qPCR-based gene expression experiments. *Bioinformatics* **29**, 461–467 (2013).
95. DePasquale, E. A. K., Schnell, D., Chetal, K. & Salomonis, N. Protocol for Identification and Removal of Doublets with DoubletDecon. *STAR Protoc* **1**, 100085 (2020).
96. Stuart, T. & Satija, R. Integrative single-cell analysis. *Nat. Rev. Genet.* **20**, 257–272 (2019).
97. Choi, Y. H. & Kim, J. K. Dissecting Cellular Heterogeneity Using Single-Cell RNA Sequencing. *Mol. Cells* **42**, 189–199 (2019).
98. Kiselev, V. Y., Andrews, T. S. & Hemberg, M. Challenges in unsupervised clustering of single-cell RNA-seq data. *Nat. Rev. Genet.* **20**, 273–282 (2019).
99. Van den Berge, K. *et al.* Trajectory-based differential expression analysis for single-cell sequencing data. *Nat. Commun.* **11**, 1–13 (2020).
100. Bai, Y.-L., Baddoo, M., Flemington, E. K., Nakhoul, H. N. & Liu, Y.-Z. Screen technical noise in single cell RNA sequencing data. *Genomics* **112**, 346–355

(2020).

101. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015).

102. Werbos, P. J. Backpropagation through time: what it does and how to do it. *Proc. IEEE* **78**, 1550–1560 (1990).

103. Kingma, D. P. & Welling, M. Auto-Encoding Variational Bayes. *arXiv [stat.ML]* (2013).

104. Arisdakessian, C., Poirion, O., Yunits, B., Zhu, X. & Garmire, L. X. DeepImpute: an accurate, fast, and scalable deep neural network method to impute single-cell RNA-seq data. *Genome Biol.* **20**, 211 (2019).

105. Badsha, M. B. *et al.* Imputation of single-cell gene expression with an autoencoder neural network. *Quantitative Biology* **8**, 78–94 (2020).

106. Deng, Y., Bao, F., Dai, Q., Wu, L. F. & Altschuler, S. J. Scalable analysis of cell-type composition from single-cell transcriptomics using deep recurrent learning. *Nat. Methods* **16**, 311–314 (2019).

107. Kingma, D. P. & Ba, J. Adam: A Method for Stochastic Optimization. *arXiv [cs.LG]* (2014).

108. Krizhevsky, A., Sutskever, I. & Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. in *Advances in Neural Information Processing Systems* (eds. Pereira, F., Burges, C. J. C., Bottou, L. & Weinberger, K. Q.) vol. 25 (Curran Associates, Inc., 2012).

109. Zhao, Z.-Q., Zheng, P., Xu, S.-T. & Wu, X. Object Detection with Deep Learning: A Review. *arXiv [cs.CV]* (2018).

110. Hicks, S. C., Townes, F. W., Teng, M. & Irizarry, R. A. Missing data and technical

variability in single-cell RNA-sequencing experiments. *Biostatistics* **19**, 562–578 (2018).

111. Quiñonero-Candela, J., Sugiyama, M., Lawrence, N. D. & Schwaighofer, A. *Dataset Shift in Machine Learning*. (MIT Press, 2009).

112. Xu, B., Wang, N., Chen, T. & Li, M. Empirical Evaluation of Rectified Activations in Convolutional Network. *arXiv [cs.LG]* (2015).

113. Clevert, D.-A., Unterthiner, T. & Hochreiter, S. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *arXiv [cs.LG]* (2015).

114. Klambauer, G., Unterthiner, T., Mayr, A. & Hochreiter, S. Self-Normalizing Neural Networks. *arXiv [cs.LG]* (2017).

115. Lu, L., Shin, Y., Su, Y. & Karniadakis, G. E. Dying ReLU and Initialization: Theory and Numerical Examples. *arXiv [stat.ML]* (2019).

116. Pascanu, R., Mikolov, T. & Bengio, Y. On the difficulty of training Recurrent Neural Networks. *arXiv [cs.LG]* (2012).

117. Talwar, D., Mongia, A., Sengupta, D. & Majumdar, A. AutoImpute: Autoencoder based imputation of single-cell RNA-seq data. *Sci. Rep.* **8**, 16329 (2018).

118. Andrews, T. S. & Hemberg, M. False signals induced by single-cell imputation. *F1000Res.* **7**, 1740 (2018).

119. Tang, F. *et al.* mRNA-Seq whole-transcriptome analysis of a single cell. *Nat. Methods* **6**, 377–382 (2009).

120. Fan, L., Zhang, F., Fan, H. & Zhang, C. Brief review of image denoising techniques. *Visual Computing for Industry, Biomedicine, and Art* **2**, 1–12 (2019).

121. Lin, C., Jain, S., Kim, H. & Bar-Joseph, Z. Using neural networks for reducing the

dimensions of single-cell RNA-Seq data. *Nucleic Acids Res.* **45**, e156–e156 (2017).

122. Köhler, N. D., Büttner, M. & Theis, F. J. Deep learning does not outperform classical machine learning for cell-type annotation. *bioRxiv* 653907 (2019) doi:10.1101/653907.

123. *Tensors in Image Processing and Computer Vision*. (Springer, London, 2009).

124. Hao, Y. *et al.* Integrated analysis of multimodal single-cell data. *bioRxiv* 2020.10.12.335331 (2020) doi:10.1101/2020.10.12.335331.

125. Paszke, A. *et al.* PyTorch: An Imperative Style, High-Performance Deep Learning Library. in *Advances in Neural Information Processing Systems* (eds. Wallach, H. et al.) vol. 32 (Curran Associates, Inc., 2019).

126. Abadi, M. *et al.* TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv [cs.DC]* (2016).

127. Fränti, P. & Sieranoja, S. How much can k-means be improved by using better initialization and repeats? *Pattern Recognit.* **93**, 95–112 (2019).

128. Reza, F. M. *An Introduction to Information Theory*. (Courier Corporation, 1994).

129. Isakova, A., Neff, N. & Quake, S. Single cell profiling of total RNA using Smart-seq-total. *bioRxiv* (2020).

130. Brockmann, L. *et al.* Molecular and functional heterogeneity of IL-10-producing CD4+ T cells. *Nat. Commun.* **9**, 5457 (2018).

131. Lake, B. B. *et al.* A comparative strategy for single-nucleus and single-cell transcriptomes confirms accuracy in predicted cell-type expression from nuclear RNA. *Sci. Rep.* **7**, 1–8 (2017).

132. Zeisel, A. *et al.* Cell types in the mouse cortex and hippocampus revealed by

single-cell RNA-seq. *Science* **347**, 1138–1142 (2015).

133. Joost, S. *et al.* Single-Cell Transcriptomics Reveals that Differentiation and Spatial Signatures Shape Epidermal and Hair Follicle Heterogeneity. *Cell Syst* **3**, 221–237.e9 (2016).

134. Stoeckius, M. *et al.* Simultaneous epitope and transcriptome measurement in single cells. *Nat. Methods* **14**, 865–868 (2017).

135. Baron, M. *et al.* A Single-Cell Transcriptomic Map of the Human and Mouse Pancreas Reveals Inter- and Intra-cell Population Structure. *Cell Syst* **3**, 346–360.e4 (2016).

136. Freytag, S., Tian, L., Lönnstedt, I., Ng, M. & Bahlo, M. Comparison of clustering tools in R for medium-sized 10x Genomics single-cell RNA-sequencing data. *F1000Res.* **7**, 1297 (2018).

137. Lake, B. B. *et al.* Neuronal subtypes and diversity revealed by single-nucleus RNA sequencing of the human brain. *Science* **352**, 1586–1590 (2016).

138. Li, H. *et al.* Reference component analysis of single-cell transcriptomes elucidates cellular heterogeneity in human colorectal tumors. *Nat. Genet.* **49**, 708–718 (2017).

139. Camp, J. G. *et al.* Multilineage communication regulates human liver bud development from pluripotency. *Nature* **546**, 533–538 (2017).

140. Romanov, R. A. *et al.* Molecular interrogation of hypothalamic organization reveals distinct dopamine neuronal subtypes. *Nat. Neurosci.* **20**, 176–188 (2016).

141. Gray Camp, J. *et al.* Human cerebral organoids recapitulate gene expression programs of fetal neocortex development. *Proc. Natl. Acad. Sci. U. S. A.* **112**, 15672–15677 (2015).

142. La Manno, G. *et al.* Molecular Diversity of Midbrain Development in Mouse, Human, and Stem Cells. *Cell* **167**, 566–580.e19 (2016).

143. Klein, A. M. *et al.* Droplet Barcoding for Single-Cell Transcriptomics Applied to Embryonic Stem Cells. *Cell* **161**, 1187–1201 (2015).

144. Usoskin, D. *et al.* Unbiased classification of sensory neuron types by large-scale single-cell RNA sequencing. *Nat. Neurosci.* **18**, 145–153 (2015).

145. Tasic, B. *et al.* Adult mouse cortical cell taxonomy revealed by single cell transcriptomics. *Nat. Neurosci.* **19**, 335–346 (2016).

146. Chen, R., Wu, X., Jiang, L. & Zhang, Y. Single-Cell RNA-Seq Reveals Hypothalamic Cell Diversity. *Cell Rep.* **18**, 3227–3241 (2017).

147. Hicks, S. C. & Peng, R. D. Elements and Principles for Characterizing Variation between Data Analyses. *arXiv [stat.AP]* (2019).

148. Johansen, N. & Quon, G. scAlign: a tool for alignment, integration, and rare cell identification from scRNA-seq data. *Genome Biol.* **20**, 1–21 (2019).

149. Langer-Safer, P. R., Levine, M. & Ward, D. C. Immunological method for mapping genes on Drosophila polytene chromosomes. *Proc. Natl. Acad. Sci. U. S. A.* **79**, 4381–4385 (1982).

150. Wang, L., Xiao, Q. & Xu, H. Optimal maximin L1-distance Latin hypercube designs based on good lattice point designs. *aos* **46**, 3741–3766 (2018).

151. Wang, L., Yang, J.-F., Lin, D. K. J. & Liu, M.-Q. NEARLY ORTHOGONAL LATIN HYPERCUBE DESIGNS FOR MANY DESIGN COLUMNS. *Stat. Sin.* **25**, 1599–1612 (2015).

152. Xiao, Q., Wang, L. & Xu, H. Application of kriging models for a drug combination

experiment on lung cancer. *Stat. Med.* **38**, 236–246 (2019).

153. Wang, L., Sun, F., Lin, D. K. J. & Liu, M.-Q. CONSTRUCTION OF ORTHOGONAL SYMMETRIC LATIN HYPERCUBE DESIGNS. *Stat. Sin.* **28**, 1503–1520 (2018).

154. Wang, L. Space-Filling Designs and Big Data Subsampling. (UCLA, 2019).

155. Wang, L. & Xu, H. A Class of Multilevel Nonregular Designs for Studying Quantitative Factors. *arXiv e-prints* arXiv:1812.05202 (2018).

156. Nishino, R. & Loomis, S. H. C. CuPy: A NumPy-compatible library for NVIDIA GPU calculations. *31st conference on neural information processing systems* 151 (2017).

157. Sun, T., Song, D., Li, W. V. & Li, J. J. scDesign2: a transparent simulator that generates high-fidelity single-cell gene expression count data with gene correlations captured. Genome Biol. 22, 1–37 (2021).

158. Song, D. & Li, J. J. PseudotimeDE: inference of differential gene expression along cell pseudotime with well-calibrated p -values from single-cell RNA sequencing data. Genome Biol. 22, 1–25 (2021).

159. Wang, L., Elmstedt, J., Wong, W. K. & Xu, H. Orthogonal Subsampling for Big Data Linear Regression. arXiv [stat.ME] (2021).