MIT Libraries | DSpace@MIT

MIT Open Access Articles

Cumulus provides cloud-based data analysis for large-scale single-cell and single-nucleus RNA-seq

The MIT Faculty has made this article openly available. ***Please share*** how this access benefits you. Your story matters.

As Published: 10.1038/S41592-020-0905-X

Publisher: Springer Science and Business Media LLC

Persistent URL: https://hdl.handle.net/1721.1/135338

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Terms of Use: Article is made available in accordance with the publisher's policy and may be subject to US copyright law. Please refer to the publisher's site for terms of use.

Massachusetts Institute of Technology

Author Manuscript

HHS Public Access
Author manuscript
Nat Methods. Author manuscript; available in PMC 2021 January 27.

Published in final edited form as:
Nat Methods. 2020 August ; 17(8): 793–798. doi:10.1038/s41592-020-0905-x.

Cumulus provides cloud-based data analysis for large-scale single-cell and single-nucleus RNA-seq

Bo Li[1,2,3,*], **Joshua Gould**[1], **Yiming Yang**[1,2], **Siranush Sarkizova**[4,5], **Marcin Tabaka**[1], **Orr Ashenberg**[1], **Yanay Rosen**[1], **Michal Slyper**[1], **Monika S Kowalczyk**[1], **Alexandra-Chloé Villani**[2,3,4,6], **Timothy Tickle**[1], **Nir Hacohen**[3,4,6], **Orit Rozenblatt-Rosen**[1,*], **Aviv Regev**[1,7,8,9,*]

[1]Klarman Cell Observatory, Broad Institute of Harvard and MIT, Cambridge, MA, USA

[2]Center for Immunology and Inflammatory Diseases, Division of Rheumatology, Allergy, and Immunology, Massachusetts General Hospital, Boston, MA, USA

[3]Department of Medicine, Harvard Medical School, Boston, MA, USA

[4]Broad Institute of Harvard and MIT, Cambridge, MA, USA

[5]Department of Biomedical Informatics, Harvard Medical School, Boston, MA, USA

[6]Center for Cancer Research, Massachusetts General Hospital, Boston, MA, USA

[7]Howard Hughes Medical Institute, Massachusetts Institute of Technology, Cambridge, MA, USA

[8]Koch Institute of Integrative Cancer Research, Massachusetts Institute of Technology, Cambridge, MA, USA

[9]Department of Biology, Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

Massively parallel single-cell and single-nucleus RNA-seq (sc/snRNA-seq) have opened the way to systematic tissue atlases in health and disease, but as the scale of data generation is growing, so does the need for computational pipelines for scaled analysis. Here, we developed Cumulus, a cloud-based framework for analyzing large scale sc/snRNA-seq datasets. Cumulus combines the power of cloud computing with improvements in algorithm implementations to achieve high scalability, low cost, user-friendliness, and integrated support for a comprehensive set of features. We benchmark Cumulus on the Human Cell Atlas Census of Immune Cells dataset of bone marrow cells and show that it substantially improves efficiency over conventional frameworks, while maintaining or improving the quality of results, enabling large-scale studies.

Users may view, print, copy, and download text and data-mine the content in such documents, for the purposes of academic research, subject always to the full Conditions of use:http://www.nature.com/authors/editorial_policies/license.html#terms

*To whom correspondence should be addressed: bli28@mgh.harvard.edu (BL), orit@broadinstitute.org (ORR), aregev@broadinstitute.org (AR).

Author Contributions
B.L. and A.R. conceived the study, designed experiments and devised analyses. B.L. developed computational methods. B.L., J.G., Y.Y. and S.S. implemented code. B.L., J.G., Y.Y., S.S., M.T., O.A and Y.R. conducted computational experiments. M.S., M.S.K. and A.V. helped interpret results from the Immune Cell Atlas data; T.T. helped with Terra cloud related development; N.H., O.R.R. and A.R. supervised work. B.L., J.G., Y.Y. and A.R. wrote the paper with input from all the authors.

Competing interests
AR is a founder of and equity holder in for Celsius Therapeutics, an SAB member of ThermoFisher Scientific, Neogene Therapeutics, and Syros Pharamceuticals, and Asimov, and an equity holder in Immunitas. NH is a founder and SAB member of Neon Therapeutics.

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Introduction

Single-cell and single-nucleus RNA-seq (sc/snRNA-seq) revolutionized our ability to study complex and heterogeneous tissues, opening the way to charting cell atlases of complex tissues in health and disease, including the Human Cell Atlas[1] and related initiatives. Advances in massively parallel sc/snRNA-seq[2,3], now allow routine profiling of millions of cells[4]. Such large and growing datasets, however, pose a significant challenge for most current analysis tools, which were designed to run on a local computer server and lack the computation capabilities required for processing terabytes of sequencing data. While some tools have been released for scRNA-seq analysis on the cloud, they supported only narrow protocols, with limited analysis scope[5]. To address this pressing challenge, we developed Cumulus, a cloud-based data analysis framework that is scalable, cost-effective, able to process a variety of data types and easily accessible to biologists.

Results

Cumulus: a cloud-based data analysis framework for single cell genomics

Cumulus consists of a cloud analysis workflow, a Python analysis package (Pegasus), and a visualization application (Cirrocumulus) (Figure 1). Cumulus performs three major steps in sc/snRNA-seq data analysis (Figure 1a): (**1**) sequence read extraction; (**2**) gene-count matrix generation; and (**3**) biological analyses. It addresses them for big sc/snRNA-seq data by combining the power of cloud computing, algorithmic improvement, and more efficient implementation. Although Cumulus is not the first attempt[5] for scRNA-seq analysis on the cloud, it is the most comprehensive one (Supplementary Table 1). In addition, we have made several technical innovations to make sure that the analysis module Pegasus works efficiently on large datasets. These innovations cover many analysis steps, such as highly variable gene selection, batch correction, inference of pseudotemporal trajectories, clustering and data visualization. To test Cumulus and compare it to other tools we relied on a scRNA-seq dataset of 274,182 cells (Methods), which were profiled from the bone marrow of 8 donors as part of the Human Cell Atlas Census of Immune Cells dataset[6].

Cumulus leverages cloud computing and compatible data platforms for the growing needs of single cell genomics. Cloud computing offers on-demand scalable computing, high-availability storage, data security and installation-free Software-as-a-Service (SaaS) capabilities, all at a low price. Cumulus is currently based on the Terra platform [https://app.terra.bio/] and Google Cloud Platform, but can be readily adapted to other cloud platforms, as it depends only on Dockers and Workflow Description Language (WDL, Methods). Cumulus executes the first two steps – sequence read extraction and gene-count matrix generation – parallelly across a large number of computer nodes, and executes the last step of analysis in a single multi-CPU node, using its highly efficient analysis module (Methods, below). Non-programming biologist users readily access computing resources on the cloud through a simple web-based user interface provided by Terra (Supplementary Video 1).

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Cumulus and Pegasus support analysis starting from a variety of input modalities

Given the growing diversity of lab protocols and experimental designed leveraging sc/snRNA-seq, we built Cumulus such that scientists can use it as a single framework for diverse data types, all of which share a single cell/nucleus transcriptome as a core readout (Supplementary Table 2). These include: droplet-based[2,7] (3' or 5' ends, with UMIs) and plate-based[8] (full length, no UMI) sc/snRNA-seq (Methods); CITE-seq[9], which simultaneously measures mRNA expression and the abundance of oligo-tagged surface antibodies in single cells (Methods), data from both cell[10] or nucleus[11] hashing experiments, which are lab techniques that reduce batch effects and cell/nucleus profiling costs, using a probabilistic demultiplexing algorithm[11] (Methods); and Perturb-seq methods for pooled CRISPR screens[12,13] with scRNA-seq readout (Methods). Other mainstream, non-Cloud based analysis packages, such as Cell Ranger[7], currently support only some of these input data types, posing a potential burden for users (Supplementary Table 2).

Pegasus, the Cumulus analysis module, supports most commonly used scRNA-seq analysis tasks, covering a comprehensive set of features (Figure 1b, Methods). Starting from a gene-count matrix, Pegasus filters out low- quality cells/nuclei, selects highly variable genes (HVG) and optionally corrects batch effects. It then performs dimensionality reduction by principal component analysis (PCA) on HVGs, constructs a *k* nearest neighbor (*k*-NN) graph on the Principal Component (PC) space, calculates diffusion maps[14,15] and applies community detection algorithms on the graph to find clusters[16,17]. It visualizes cell profiles using either t-SNE[18,19]-based or UMAP[20,21]-based methods. It can additionally estimate diffusion pseudotime[15] and visualize developmental trajectories using force-directed layout embedding (FLE)[22–24] based algorithms. Pegasus can be used to detect cluster-specific markers, by differential expression analysis between cells within and outside of a cluster and optionally calculates the area under ROC curve (AUROC) values for all genes (Methods). It can also train a gradient boosting tree classifier[25] on the gene expression matrix to predict cluster labels and output genes with high feature importance scores (Methods), which provide additional information for detecting cluster-specific markers. Lastly, it annotates clusters with putative cell type labels based on user-provided gene sets (Methods). Pegasus thus offers diverse features, comparable to two other mainstream packages, Seurat[26] and SCANPY[27], although each package also has some unique features, absent from the other two (Supplementary Table 3). Pegasus can run on the cloud or as an independent Python package.

Interactive data visualization, *post hoc* analysis and data deposition in the Cumulus ecosystem

Cumulus supports additional tasks using Cirrocumulus and scPlot, as well as through seamless connection to third party resources. First, once the data are analyzed, users can visualize their results instantly using Cirrocumulus, a serverless application that enables interactive data visualization and sharing (Figure 1c, Supplementary Video 2). Since Cirrocumulus only downloads to the browser those data that are necessary for visualization (Methods), it is scalable to millions of cells. Moreover, users can also download Cumulus-produced HDF5 result files for use with other visualization tools such as cellxgene [https://github.com/chanzuckerberg/cellxgene], UCSC Cell Browser [http://cells.ucsc.edu/], and

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

scSVA[24] (Figure 1c). Alternatively, users can inspect or re-analyze their data interactively using Pegasus on Terra Jupyter Notebooks (Figure 1c, Supplementary Video 3). To help users better navigate data, we developed scPlot (Methods), a python package for generating interactive figures, as part of Pegasus. Finally, users can synchronize Cumulus results to the Single Cell Portal [https://singlecell.broadinstitute.org/single_cell] for data deposition (Figure 1c). To help users adopt Cumulus, it is demoed as a featured workspace on Terra [https://app.terra.bio/#workspaces/kco-tech/Cumulus].

Enhanced scalability of key analysis tasks

We enhanced the performance of the Pegasus analysis module through several algorithmic and implementation improvements in some of the most intensive tasks: the selection of highly variable genes (HVGs), batch correction, *k*-NN graph construction, calculation of diffusion pseudotime (DPT), a combination of spectral and community-based clustering, and efficient visualization algorithms. We describe each of these enhancements in turn, comparing its impact on analysis quality and analysis speed/scale with other major packages. These enhancements are further described below.

First, we implemented a new HVG selection procedure that simplifies the calculation process and provides a mathematically sound way to handle batch effects (Extended Data Figure 1a, Methods). For users' convenience, we also include a standard procedure[26], which is used by both SCANPY and Seurat. Comparing the new and standard procedures when applied to the bone marrow dataset suggests that the new procedure has at least equal quality *vs.* the standard one. It recovers slightly more immune-specific genes provided by the ImmPort[28] data repository (Extended Data Figure 1b, Supplementary Data 1, Methods), including important T cell markers, such as *CD3D*, *CD3E* and *CD4*, which are missed by the standard procedure. It also identifies one more cell type, megakaryocytes, which is missed by the standard procedure (Extended Data Figure 1c).

Next, we enhanced the scalability of batch correction in Pegasus, by implementing the classical location and scale (L/S) adjustment method[29], which relies only on linear operations and is thus much faster. As a benchmark, we ran Pegasus's L/S method, SCANPY's offerings of Combat[30], MNN[31] and BBKNN[32], and the most recent integration method[33] of Seurat v3 on a subset of the bone marrow dataset (Methods) and compared each method's batch correction efficiency using two measurements, the kBET[34] and kSIM acceptance rates. Briefly, the kBET acceptance rate measures if batches are well-mixed in the local neighborhood of each cell; the kSIM acceptance rate measures if cells of the same pre-annotated cell type are still close to each other in the local neighborhoods after batch correction (Methods), and helps reflect if known biological relations are preserved after correction. An ideal batch correction method should have both high kBET and kSIM acceptance rates. Each of the five methods evaluated showed a trade-off between the two rates, and none was a clear best performer (Figure 2a), Pegasus's L/S method is the fastest (Extended Data Figure 2a), while maintaining a good balance between the two rates (Figure 2a, Extended Data Figure 2b–g).

We also enhanced the scalability of *k*-NN graph construction by adopting the Hierarchical Navigable Small World (HNSW)[35] algorithm, a state-of-the-art approximate nearest-

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

neighbor finding algorithm, which was previously shown to be fastest for high quality approximations[36]. We compared HNSW with the approximate nearest neighbor finding algorithms used by SCANPY and Seurat on the bone marrow dataset based on speed and on recall, defined as the percentage of nearest neighbors that are also found by the brute-force algorithm (Methods). HNSW has a near optimal recall (Extended Data Figure 3a), while being 3–19x faster (Extended Data Figure 3b). HNSW was also benchmarked recently for plate based, small scale, scRNA-seq[37].

To speed up the calculation of diffusion maps[14] and diffusion pseudotime[15] (DPT), we adopted two modifications, also used by SCANPY and scSVA, and further improved the identification of pseudotemporal trajectories[15] with two additional modifications. As in SCANPY, we construct the affinity matrix based on the approximate k-NN graph constructed in the previous step instead of a complete graph, and also use only the top n diffusion components to approximate diffusion distances and thus diffusion pseudotimes, where n is a user-specific parameter. In addition, to better identify pseudotemporal trajectories when there are multiple subsets of cells undergoing separate temporal processes, we found that using more diffusion components helps us better separate different cell populations (Extended Data Figure 4a, red regions) and thus we set $n = 100$ by default. We further introduce a family of diffusion pseudotime maps parameterized by timescale t: each pseudotime "meta-map" is constructed by summing over diffusion maps up to its timescale t (Methods). The DPT method[15] is equivalent to the special case of this family with $t = \infty$. As timescale t increases, diffusion maps begin to smooth out local noise[38]. However, when t becomes too large, diffusion maps will also smooth out real signals[38]. Thus, instead of $t = \infty$, we choose a timescale t that smooths out most noise but little signal. Inspired by the PHATE[39] method, we propose to pick t as the knee point on the curve of von Neumann entropies[40] induced by diffusion maps at different timescales (Extended Data Figure 4b, Methods). In the Immune Cell Atlas data, using the selected t, we identify a trajectory (Figure 2b) that more clearly bifurcates from hematopoietic stem cells into *CD14*$^+$ monocytes and conventional dendritic cells (cDCs), whereas those two lineages overlap in the DPT model.

Spectral community detection algorithms combine speed and quality in clustering

In addition to offering popular modularity-based community detection algorithms for clustering cell profiles, including Louvain[16] and Leiden[17] (Methods), Pegasus also includes spectral-community-detection algorithms, such as spectral-Louvain and spectral-Leiden, which combine the strengths of both spectral clustering[41] and community detection algorithms (Methods). Spectral clustering performed by applying the k-means algorithm on the calculated diffusion pseudotime components is very fast, but its clustering results are not always satisfactory (Extended Data Figure 5a). Conversely, in Pegasus's spectral clustering, we first aggregate cells into small groups and then apply community detection algorithms on the aggregated groups (Methods). On the bone marrow dataset, this new method provides clusters that are comparable to those from modularity-based community detection algorithms, but at the high speed of spectral clustering (Extended Data Figure 5b,c).

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

A deep learning visualization technique speeds up common visualization modalities

In addition to visualization of single cell profiles using either t-SNE[18], FIt-SNE[19], UMAP[20] (Methods), or a force directed layout embedding (FLE[23]) of the diffusion pseudotime map (Methods), we also include a deep-learning-based visualization technique that speeds up a generalized set of these and similar visualization algorithms (Methods). Inspired by net-SNE[42], this technique is based on the assumption that large datasets are often redundant and their global structure can be captured using only a portion of the data. It thus first subsamples a fraction of cells according to each cell's local density, ensuring higher rate of sampling from rare and sparse clusters, and then embeds the subsampled cells using the embedding algorithm of interest, such as UMAP (Figure 2c). It then trains a deep-learning-based regressor (Methods) on the subsampled cells using their embedded coordinates as ground truth and uses the regressor to predict embedding coordinates for the remaining cells. Because the predicted coordinates yield a "blurry" visualization (Figure 2c), it has a final refinement step, which applies the embedding algorithm for a small number of iterations to both the calculated coordinates for the subsampled cells and predicted coordinates for the remaining cells (Figure 2c). We call visualizations obtained using this technique as Net-* visualizations and show that they speed up the original embedding algorithm by at least 2x and maintain the visualization quality based on similar kSIM acceptance rates (Figure 2d and Extended Data Figure 6).

Benchmarking Pegasus and alternative software packages

As a result of the combined algorithmic and implementation improvements, Pegasus is much faster than other packages for running key analyses tasks on the bone marrow dataset[6] (Supplementary Table 4, Methods) and the 1.3 million mouse brain dataset from 10x Genomics [https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.3.0/1M_neurons] (Supplementary Table 5, Methods). We validated that these improvements run efficiently on both small and large data sets with different thread settings by subsampling the bone marrow dataset (Supplementary Data 2, Methods) and demonstrated that Pegasus works equally well for small datasets on a 10x Genomics 5K PBMC dataset[https://support.10xgenomics.com/single-cell-gene-expression/datasets/3.0.2/5k_pbmc_v3] (Supplementary Table 6, Methods). Furthermore, Pegasus is memory efficient compared to other packages, based on peak memory use on the 5K PBMC and bone marrow datasets (Supplementary Table 7, Methods).

Benchmarking the cloud-based framework of Cumulus

With its cloud-based architecture, Cumulus is much faster than alternatives when benchmarking on the bone marrow dataset. Compared to a Cell Ranger + Seurat/SCANPY pipeline, Cumulus completed the analysis in around 15 hours, while the alternative pipeline took over 9 days to run (Table 1, Methods). The associated computational costs were modest (*e.g.*, ~$2 on average for around 4,000 cells in one sample, Table 1, Methods). Because the *count* step showed the most speedup gain, we further validated that this step is time and cost-efficient for both small and large datasets (Extended Data Figure 7, Supplementary Data 3, Methods). In addition, since Cumulus utilizes Docker, the gene-count matrices generated on the cloud and on the local server are identical for both droplet-based and plate-based

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

scRNA-seq data (Supplementary Data 4, Methods). Besides Cell Ranger, Cumulus also supports gene-count matrix generation using Optimus (the HCA pipeline, https://data.humancellatlas.org/pipelines/optimus-workflow), Alevin[43], Kallisto-BUStools[44], or STARsolo [https://github.com/alexdobin/STAR/blob/2.7.3a/docs/STARsolo.md] and we have benchmarked cloud-based performance of Cell Ranger, Optimus, Alevin, Kallisto-BUStools and STARsolo on the 5K PBMC dataset (Supplementary Table 8, Methods).

Discussion

Cumulus provides the community with a cloud-based, scalable, cost-effective, comprehensive and easy to use platform for single-cell and single-nucleus RNA-seq research. Pegasus, Cumulus' analysis module, which can also be used as an independent Python package, implements many improvements which enhance efficiency, from a new HVG selection procedure to a generalized deep-learning-based visualization speedup. While a complex framework such as Cumulus cannot provide an optimal combination of specialized algorithms for each application, it will accelerate research by providing an integrated and fast approach that enables more labs to analyze large-scale single-cell and single-nucleus datasets. In a separate study[45], we demonstrated Cumulus by analyzing single-cell and single-nucleus RNA-seq data from fresh and frozen tumor from the Human Tumor Atlas Pilot Project (HTAPP). As the community produces data sets at substantially larger scales, we hope that Cumulus will play an important role in the effort to build atlases of complex tissues and organs at higher cellular resolution and leveraging them to understand the human body in health and disease.

Online Methods

CUMULUS MODULES

Gene-count matrix generation for droplet-based scRNA-seq—Cumulus supports gene-count matrix generation for 10x Genomics V2 and V3 chemistry using Cell Ranger. Cumulus first demultiplexes Illumina base call files (BCLs) for each sequencing flowcell by running *mkfastq* steps parallelly in different computing nodes. Each *mkfastq* job calls 'cellranger mkfastq' to generate sequence reads in FASTQ files. By default, each *mkfastq* job requests 32 CPUs, 120 GB memory and 1.5 TB disk space from the cloud. Cumulus then generates gene-count matrices for each 10x channel by running *count* steps in parallel. Each *count* job calls 'cellranger count' with appropriate parameters and requests 32 CPUs, 120 GB memory and 500 GB disk space from the cloud by default.

Alternatively, Cumulus can run *count* jobs using Optimus, Alevin, Kallisto-BUStools, or STARsolo. For Alevin, Kallisto-BUStools and STARsolo, we request 32 CPUs, 120 GB memory and 500 GB disk space from the cloud by default. For Optimus, we request computing resources using its default parameters. One key task in the *count* job is to align sequence reads to a reference genome and there are several Apache Spark-based tools[46] that enable read alignment using multiple nodes. However, since the sequencing data produced from each 10x Genomics channel fit into one multi-CPU node perfectly, using Spark-based tools would be excessive. Cumulus also supports gene-count matrix generation for Drop-

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

seq[2] data using either the methods described in Drop-seq alignment cookbook [https://github.com/broadinstitute/Drop-seq] or dropEst[47].

Gene-count matrix generation for plate-based scRNA-seq—Cumulus supports gene-count matrix generation for scRNA-seq data generated by the SMART-seq2 protocol[8] from sequence reads in FASTQ files. Cumulus estimates gene expression levels for each single cell in parallel in different computer nodes. Each node runs RSEM[48] with default parameters and utilizes HISAT2[49] with the Human Cell Atlas SMART-seq2 pipeline [https://data.humancellatlas.org/pipelines/smart-seq2-workflow] parameters to align reads to a reference transcriptome. Cumulus also offers options to align reads to a reference transcriptome using Bowtie 2[50] or to a reference genome using STAR[51]. Each node requests 4 CPUs and 3.6 GB memory by default, if utilizing HISAT2 or Bowtie 2 aligner and requests 4 CPUs and 32 GB memory by default if utilizing STAR aligner. Each node requests an appropriate disk space calculated based on reference and read file sizes. Once expression levels are estimated, Cumulus converts the relative expression levels (in Transcript per 100K, TP100K) into a count vector for each single cell using the formula below and then generates a gene-count matrix by concatenating count vectors from all cells.

$$c_i = \left\lfloor \frac{TP100K_i}{10^5} \cdot c_{tot} \right\rfloor, \quad (1)$$

where c_i and $TP100K_i$ are the converted read count and estimated expression level of gene i, respectively. c_{tot} is the sum of RSEM-estimated expected counts from all genes.

Pegasus—Cumulus runs the *analysis* step on a single node, which requests 32 CPUs, 200 GB memory and 100 GB disk space by default. The *analysis* step calls Pegasus, a fast Python package we have implemented. Pegasus utilizes SCANPY's AnnData data structure[27] to store gene-count matrices and analysis results. More implementation details are discussed in the subsequent sections and Supplementary Note 1.

Feature-count matrix generation for CITE-seq, cell hashing, nucleus hashing, and Perturb-seq—Cumulus supports feature-count matrix generation of CITE-seq[9], cell hashing[10], nucleus[11] hashing and Perturb-seq[12,13] protocols, using either 10x Genomics V2 or V3 chemistry. Each feature-count matrix generation job runs parallelly on a separate compute node with 1 CPU, 32 GB memory and 100 GB disk space, and calls 'generate_count_matrix_ADTs', a fast C++ program we implemented, to extract the matrix from sequence reads in FASTQ files. The C++ program scans each read pair to search for valid sequence structures. We assume read 1 records the cellular barcode and Unique Molecular Identifier (UMI) information and read 2 records feature barcode information, such as hash tags for hashing protocols or sgRNA information for Perturb-seq (below). The first 16 nucleotides of read 1 represent the cell barcode for both V2 and V3 chemistry. The next 10 and 12 nucleotides represent the UMI for V2 and V3 chemistry, respectively. We allow up to 1 and 0 mismatch for matching cell barcodes in V2 and V3 chemistry, respectively.

Author Manuscript
Author Manuscript
Author Manuscript
Author Manuscript

Feature barcode information is recorded differently in read 2 for different protocols. For CITE-seq, cell hashing and nucleus hashing protocols, the location of the feature barcode depends on what type of BioLegend TotalSeq™ antibodies users choose. If TotalSeq™-A antibodies are used, the feature barcode is located at the 5' end of read 2. Otherwise, the feature barcode starts at the 11th nucleotide from the 5' end of read 2. 'generate_count_matrix_ADTs' automatically detects antibody type by scanning read 2 of the first 1,000 read pairs and calculating the percentage of read pairs containing the auxiliary sequence. If more than 50% of read pairs contain the auxiliary sequence, we assume the antibody type is TotalSeq™-A, otherwise it is TotalSeq™-B or TotalSeq™-C. We allow up to 1 mismatch for matching the auxiliary sequence.

For Perturb-seq protocols, we assume that the feature barcode (protospacer) is located in front of a user-provided anchor sequence. We first search the anchor sequence in read 2, allowing up to 2 mismatches or indels. We then extract the feature barcode at the 5' end of the anchor sequence.

Once we locate the feature barcode, we match it with a user-provided white list, allowing up to 3 mismatches by default. After scanning all read pairs, 'generate_count_matrix_ADTs' generates a feature-count matrix in CSV format: each row represents one feature, each column represents one cell barcode, and each element records the number of unique UMIs for the feature in the row in the cell barcode in the column. To speed up sequence matching, we encode cell barcodes, UMIs and feature barcodes into 8-byte unsigned integers (2 bits per nucleotide).

CITE-seq data analysis—Based on the generated feature-count matrix, Cumulus first calculates the log fold change between feature UMI counts of the antibody of interest and its IgG control as the antibody expression, provided that users include both antibodies of interest and their corresponding IgG controls in their CITE-seq assays. Let us denote the UMI counts of the antibody and its IgG control as c_a and c_c. The antibody expression *expr* is calculated as

$$expr = \max(\log \frac{c_a + 1}{c_c + 1}, 0), \tag{2}$$

where we add 1 to both the numerator and denominator to avoid log 0. If IgG controls are not provided, we calculate *expr* as

$$expr = \log(c_a + 1). \tag{3}$$

Cumulus merges the transformed antibody expression matrix into an RNA expression matrix so that users can plot antibody expression in 2D visualizations (*e.g.*, t-SNE & UMAP) calculated based on RNA expression levels. Cumulus can optionally generate t-SNE plots solely based on antibody expression levels.

Demultiplexing cell hashing and nucleus hashing data—Cumulus demultiplexes cell hashing and nucleus hashing data using the DemuxEM algorithm, which we recently described[11].

Chimeric read filtration for Perturb-seq data—In Perturb-seq, sgRNAs are often amplified by dial-out PCR[12] to ensure feature detection, and the resulting library is often over-sequenced, which can lead to a high number of false positive UMIs due to PCR chimeric reads[52]. Such false positive UMIs tend to have fewer supporting reads on average. Suppose we have $c[i]$ UMIs with exact i supporting reads. In general, we expect $c[i]$ to decrease monotonically as i increases. However, if the library is over-sequenced, we may observe a second peak in the tail of the $c[i]$ distribution ($\exists i, c[i-2] < c[i-1] < c[i] \quad c[i+1]$), which is more likely to represent true UMIs. Cumulus detects the left boundary of the second peak by scanning i consecutively. If Cumulus can find an i such that $c[i] < c[i+1] < c[i+2]$ and $i \quad 10$, Cumulus will filter out any UMIs with fewer than i supporting reads. Otherwise, Cumulus filters out any UMIs with only one supporting read. If a cell barcode and UMI combination contains more than 1 feature barcode, it is likely that the feature barcode with fewer supporting reads is produced by PCR chimeras[52] and Cumulus will filter feature barcodes supported by no more than 10% of reads belonging to that combination. Cumulus generates a filtered feature-count matrix after this filtration step and lets users decide if they want to use the original feature-count matrix or the filtered feature-count matrix.

Computational load balancing—Balancing the computational load is an important task for designing parallel systems. For analyzing droplet-based data, we partition the analysis workflow into three major steps: *mkfastq*, *count* and *analysis*. The *mkfastq* step extract reads from each Illumina flow cell in different nodes parallelly. In most cases, users sequence their libraries using one type of sequencer (*e.g.*, NextSeq, HiSeq, or NovaSeq), and flow cells from the same type of sequencer often deliver similar sequencing throughputs. Thus, the data load in the *mkfastq* step is balanced naturally. The *count* step generates a gene-count matrix for each channel in parallel using different nodes. When experimentalists design their scRNA-seq experiments, they aim to sequence each channel to comparable sequencing depth. Thus, the computational load is usually balanced naturally, due to experimental design. The *analysis* step gathers gene-count matrices from all samples and runs downstream analyses on a single multi-CPU node and thus data load balancing is not relevant. For analyzing plate-based SMART-seq2 data, we generate an expression vector for each single cell in parallel across nodes and then pool the estimated expression vectors together as the gene-count matrix. In a good quality SMART-seq2 library, the number of transcripts from each single cell should be comparable and thus the data load is balanced. In less-than-ideal libraries, some cells may capture many more reads than other cells. However, for SMART-Seq2 data, single cells typically have less than 2M reads, which can be analyzed very quickly. Thus, for SMART-Seq2 data, we did not invest additional efforts optimizing the data load balance.

Cirrocumulus implementation—Cirrocumulus is an interactive web application for exploring single cell datasets. It can be hosted on Google App Engine application for

Author Manuscript
Author Manuscript
Author Manuscript
Author Manuscript

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

collaborative use or can be run in standalone mode on a personal computer. Cirrocumulus consists of a client-side component implemented in JavaScript and a server component implemented in Python. The client uses React to manage state and WebGL to visualize variables on a 2D or 3D embedding in a performant manner. The server component consists of functions to manage datasets, slice variables from a dataset stored as a folder of JSON and PARQUET files, and can optionally generate statistical summaries on an n-dimensional grid, thus enabling plotting of millions of cells. Cumulus has options to generate the Cirrocumulus input folder automatically for users. In the standalone mode, Cirrocumulus can additionally support datasets in AnnData or Zarr formats.

scPlot implementation—scPlot [https://github.com/klarman-cell-observatory/scPlot] is a plotting library included as part of Pegasus. Plots provided include scatter plots, feature plots, dot plots and violin plots and can scale to millions of cells by plotting cells on a two-dimensional grid. scPlot uses HoloViews [http://holoviews.org/], thus allowing the same code to generate interactive plots with Bokeh for a Jupyter notebook and static plots with Matplotlib.

Preprocessing—Pegasus selects high quality cells based on a combination of the following criteria, with user-provided parameters: 1) number of unique molecular identifiers (UMIs) between $[min_umis, max_umis)$, default: $min_umis = 100$ and $max_umis = 600{,}000$; 2) number of expressed genes (at least one UMI) between $[min_genes, max_genes)$, default: $min_genes = 500$ and $max_genes = 6000$; 3) percentage of UMIs from mitochondrial genes less than $percent_mito$, default: $percent_mito = 10\%$. Pegasus then selects robust genes, defined as genes detected in at least x percentage of cells, where x is a user-defined parameter; default: $x = 0.05\%$ (equivalent to 3 cells out of 6,000 cells). Next, Pegasus normalizes the count vector of each cell, such that the sum of normalized counts from robust genes is equal to 100,000 transcripts per 100K (TP100K), and transforms the normalized expression matrix into the natural log space by replacing expression value y into $\log(y + 1)$. Additional details are available in Supplementary Note 1.

Highly variable gene selection—The standard HVG selection procedure operates in the original expression space. However, almost all downstream analyses are conducted in log expression space. To reconcile this inconsistency, we develop a new HVG selection procedure that operates directly in log expression space.

We select HVGs only from robust genes. Suppose we have N cells and R robust genes. We denote the log expression of gene g in cell i as Y_{ig}. We first estimate the mean and variance for each robust gene g as

$$\hat{\mu}_g = \frac{1}{N}\sum_{i=1}^{N} Y_{ig} \quad and \quad \hat{\sigma}_g^2 = \frac{1}{N-1}\sum_{i=1}^{N} (Y_{ig} - \hat{\mu}_g)^2. \tag{4}$$

We then fit a LOESS[53] curve of degree 2 (span parameter 0.02) between the estimated means and variances (Extended Data Figure 1a) and denote the LOESS-predicted variance for gene g as $\tilde{\sigma}_g^2$. Any gene g with $\hat{\sigma}_g^2 > \tilde{\sigma}_g^2$ has a higher than expected variance.

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

We calculate the difference and fold change between the estimated and LOESS-predicted variances as

$$\sigma_g = \hat{\sigma}_g^2 - \tilde{\sigma}_g^2 \quad \text{and} \quad \tau_g = \frac{\hat{\sigma}_g^2}{\tilde{\sigma}_g^2}. \tag{5}$$

We then rank each robust gene with respect to σ_g and τ_g in descending order, and denote their rankings as $rank_\sigma(g)$ and $rank_\tau(g)$ respectively. Lastly, we define the overall ranking as the sum of the two rankings

$$rank(g) = rank_\sigma(g) + rank_\tau(g), \tag{6}$$

and select the top n robust genes with respect to $rank(g)$ as HVGs.

The new procedure handles batch effects naturally. Suppose we have K biologically different groups, each group k has n_k batches and each batch kj has n_{kj} cells. We additionally denote the mean within batch kj and within group k as $\hat{\mu}_{kjg}$ and $\hat{\mu}_{kg}$, respectively. Because we have

$$\sum_{i=1}^{n_{kj}} (Y_{kjig} - \hat{\mu}_{kjg}) = 0 \quad and \quad \sum_{j=1}^{n_k} n_{kj}(\hat{\mu}_{kjg} - \hat{\mu}_{kg}) = 0, \tag{7}$$

We can decompose the variance $\hat{\sigma}_g^2$ into three components – within-batch variance ($\hat{\sigma}_{g1}^2$), between-batch variance ($\hat{\sigma}_{g2}^2$) and between-group variance ($\hat{\sigma}_{g3}^2$) – as follows:

$$\begin{aligned}
\hat{\sigma}_g^2 &= \frac{1}{N-1}\sum_{i=1}^{N}(Y_{ig} - \hat{\mu}_g)^2 \\
&= \frac{1}{N-1}\sum_{k=1}^{K}\sum_{j=1}^{n_k}\sum_{i=1}^{n_{kj}}((Y_{kjig} - \hat{\mu}_{kjg}) + (\hat{\mu}_{kjg} - \hat{\mu}_{kg}) + (\hat{\mu}_{kg} - \hat{\mu}_g))^2 \\
&= \frac{1}{N-1}\sum_{k=1}^{K}\sum_{j=1}^{n_k}\sum_{i=1}^{n_{kj}}(Y_{kjig} - \hat{\mu}_{kjg})^2 + \frac{1}{N-1}\sum_{k=1}^{K}\sum_{j=1}^{n_k} n_{kj}(\hat{\mu}_{kjg} - \hat{\mu}_{kg})^2 \\
&+ \frac{1}{N-1}\sum_{k=1}^{K}(\sum_{j=1}^{k} n_{kj})(\hat{\mu}_{kg} - \hat{\mu}_g)^2 \\
&= \hat{\sigma}_{g1}^2 + \hat{\sigma}_{g2}^2 + \hat{\sigma}_{g3}^2.
\end{aligned} \tag{8}$$

We remove the variance term ($\hat{\sigma}_{g2}^2$) due to batch effects by redefining $\hat{\sigma}_g^2$ as

$$\hat{\sigma}_g^2 := \hat{\sigma}_{g1}^2 + \hat{\sigma}_{g3}^2, \tag{9}$$

and plug in the new redefined variance term to the previously described procedure to select HVGs. If we use the redefined variance terms in HVG selection, we are in the batch-aware mode. We provide a detailed description of the new procedure in the Supplementary Note 1 1.

Author Manuscript
Author Manuscript
Author Manuscript
Author Manuscript

We also implemented the standard HVG selection procedure, which handles batch effects using the method in Seurat v3[33], and documented implementation details in the Supplementary Note 1.

Batch correction with the L/S adjustment method—For simplicity, let us assume that we only have one biological group with m batches and each batch j has n_j cells. We model the log gene expression level of gene g at batch j's ith cell as

$$Y_{jig} = \alpha_g + \gamma_{jg} + \delta_{jg}\epsilon_{jig}, \tag{10}$$

where α_g is the baseline expression level of gene g, ϵ_{jig} is the error term, which follows a distribution with a mean 0 and a variance σ_g^2. In addition, γ_{jg} and δ_{jg} are the additive and multiplicative batch effects, respectively. We estimate these parameters for each gene separately as follows:

$$\hat{\alpha}_g = \frac{1}{N}\sum_{j=1}^{m}\sum_{i=1}^{n_j} Y_{jig}, \tag{11}$$

$$\hat{\gamma}_{jg} = \frac{1}{n_j}\sum_{i=1}^{n_j} Y_{jig} - \hat{\alpha}_g, \tag{12}$$

$$\hat{\sigma}_g = \sqrt{\frac{1}{N}\sum_{j=1}^{m}\sum_{i=1}^{n_j}\left(Y_{jig} - \hat{\alpha}_g - \hat{\gamma}_{jg}\right)^2}, \tag{13}$$

$$\hat{\delta}_{jg} = \frac{\sqrt{\frac{1}{n_j - 1}\sum_{i=1}^{n_j}\left(Y_{jig} - \hat{\alpha}_g - \hat{\gamma}_{jg}\right)^2}}{\hat{\sigma}_g}. \tag{14}$$

We denote Y^*_{jig} as the batch adjusted expression level, which is calculated as

$$Y^*_{jig} = \frac{Y_{jig} - \hat{\alpha}_g - \hat{\gamma}_{jg}}{\hat{\delta}_{jg}} + \hat{\alpha}_g. \tag{15}$$

We provide a more detailed description of the L/S method in the Supplementary Note 1, including how to handle multiple biological groups.

Since batch correction transforms a sparse expression matrix into a dense matrix, which uses much more memory, we only calculate batch-adjusted expression levels for genes of interest, such as HVGs. We rewrite (15) as

$$Y_{jig}^{*} = \frac{1}{\hat{\delta}_{jg}} Y_{jig} + \left(\hat{\alpha}_g - \frac{\hat{\alpha}_g + \hat{\gamma}_{jg}}{\hat{\delta}_{jg}}\right), \tag{16}$$

and use a two-step procedure to correct batch effects: First, we calculate and save batch-correction factors $\frac{1}{\hat{\delta}_{jg}}$ and $\hat{\alpha}_g - \frac{\hat{\alpha}_g + \hat{\gamma}_{jg}}{\hat{\delta}_{jg}}$ for all genes. Second, we calculate adjusted expression levels only for genes of interest using (16). We save the batch-correction factors for all genes, such that we can calculate batch-adjusted expression levels for any gene instantly in the future.

kBET acceptance rate—kBET[34] acceptance rate measures if cells from different batches mix well in the local neighborhood of each cell. Pegasus implements the kBET acceptance rate calculation procedure as follows: We define $f = (f_1, \ldots, f_m)$ as the ideal batch mixing frequency, where $f_j = \frac{n_j}{N}$. For each cell i, we find its k nearest neighbors (including itself) using the HNSW algorithm[35] and denote the number of neighbors belonging to batch j as n_{ji}^k. Then we calculate its χ^2 test statistic with $m - 1$ degrees of freedom as

$$\kappa_i^k = \sum_{j=1}^{m} \frac{\left(n_{ji}^k - f_j \cdot k\right)^2}{f_j \cdot k}, \tag{17}$$

and its p value as

$$p_i^k = 1 - F_{m-1}\left(\kappa_i^k\right), \tag{18}$$

where $F_{m-1}(x)$ is the cumulative density function.

The kBET acceptance rate is calculated as the percentage of cells that accept the null hypothesis at significance level α:

$$kBET\ rate = \frac{\sum_{i=1}^{N} I\left(p_i^k \geq \alpha\right)}{N} \times 100\%, \tag{19}$$

where $I(x)$ is the indicator function, and k and α are user-specified parameters.

kSIM acceptance rate—The kSIM acceptance rate requires ground truth cell type information and measures if the neighbors of a cell has the same cell type as it does. If a method over-corrects the batch effects, it will have a low kSIM acceptance rate. We use the HNSW algorithm to find k nearest neighbors (including the cell itself) for each cell i and denote the number of neighbors that have the same cell type as i as n_i^k. In addition, we require at least β fraction of neighbors of cell i to have the same cell type as i in order to say cell i has a consistent neighborhood. The kSIM acceptance rate is calculated as follows:

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

$$kSIM\ rate = \frac{\sum_{i=1}^{N} I(\frac{n_i^k}{k} \geq \beta)}{N} \times 100\%, \quad (20)$$

where k and β are user-specified parameters.

Dimensionality reduction by Principal Components Analysis—Pegasus calculates the top m principal components based on highly variable genes. It utilizes the randomized PCA algorithm[54] implemented in *Scikit-learn* package[55] to speed up the computation. By default, Pegasus sets $m = 50$.

***k*-nearest neighbors (*k*-NN) graph construction—**Pegasus uses the HNSW[35] algorithm with parameters $M = 20, efC = 200$, $efS = 200$, to construct k-NN graphs. By default, Pegasus searches the top 100 nearest neighbors (including the cell itself) for each cell ($K = 100$). Because HNSW is an approximate algorithm, it cannot always return the cell itself as the 1st nearest neighbor. For any cell missing itself as the 1st nearest neighbor, Pegasus sets itself as the 1st nearest neighbor and picks the top 99 nearest neighbors returned by HNSW as the 2nd to 100th nearest neighbors. HNSW has a random index building process, which produces different indices in different runs if multiple threads are used. For reproducibility purposes, Pegasus provides two modes of running HNSW: robust mode and full speed mode. In robust mode, Pegasus runs the index building process with only one thread and runs the neighbor searching process with multiple threads. In full speed mode, Pegasus also runs the index building process with multiple threads. In either mode, Pegasus stores the neighbor searching results in the AnnData[27] object. Without explicit notification, Pegasus runs HNSW in the robust mode.

Diffusion maps and diffusion pseudotime maps—We provide a high-level summary here and a more detailed description in Supplementary Note 1.

To compute diffusion maps, we first construct an affinity matrix $\mathbf{W}_{N \times N}$ based on the top m principal components. This affinity matrix is also used in community-detection-based clustering algorithms. We construct $\mathbf{W}$ based on the top K nearest neighbors found by the HNSW algorithm. Let us define a cell $\mathbf{x}$'s neighborhood set $N(\mathbf{x})$ as the set consisting of $\mathbf{x}$'s 2nd to Kth nearest neighbors. We then define the following locally scaled Gaussian kernel between any two cells $\mathbf{x}$ and :

$$K(\mathbf{x}, \mathbf{y}) = \left(\frac{2\sigma_{\mathbf{x}}\sigma_{\mathbf{y}}}{\sigma_{\mathbf{x}}^2 + \sigma_{\mathbf{y}}^2}\right)^{\frac{1}{2}} \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{\sigma_{\mathbf{x}}^2 + \sigma_{\mathbf{y}}^2}\right), \quad (21)$$

where, $\mathbf{x}$ is a vector containing the top m PC coordinates of cell $\mathbf{x}$, and $\sigma_{\mathbf{x}}$ is $\mathbf{x}$'s local kernel width, defined as $\sigma_{\mathbf{x}} = \text{median}\{d_i | i = 2, \ldots, K\}$, where d_i is the distance between cell $\mathbf{x}$ and its ith neighbor. To eliminate the effects of sampling density, we additionally define the following density-normalized kernel[14]:

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

$$K'(\mathbf{x}, \mathbf{y}) = \frac{K(\mathbf{x}, \mathbf{y})}{q(\mathbf{x})q(\mathbf{y})}, \tag{22}$$

where $q(\mathbf{x})$ is the sampling density term of cell $\mathbf{x}$ and defined as:

$$q(\mathbf{x}) = \sum_{\mathbf{y} \in N(\mathbf{x}) \text{ or } \mathbf{x} \in N(\mathbf{y})} K(\mathbf{x}, \mathbf{y}). \tag{23}$$

The affinity matrix $\mathbf{W}$ is constructed using the density-normalized kernel:

$$\mathbf{W}(\mathbf{x}, \mathbf{y}) = \begin{cases} K'(\mathbf{x}, \mathbf{y}), & \mathbf{y} \in N(\mathbf{x}) \text{ or } \mathbf{x} \in N(\mathbf{y}) \\ 0, & otherwise \end{cases}. \tag{24}$$

We then calculate the Markov chain transition matrix $\mathbf{P}$ and the symmetric "transition" matrix $\mathbf{Q}$ based on the affinity matrix:

$$\mathbf{P} = \mathbf{D}^{-1}\mathbf{W}, \qquad \mathbf{D} = diag\left(\sum_{\mathbf{y}} \mathbf{W}(\mathbf{x}, \mathbf{y})\right), \tag{25}$$

$$\mathbf{Q} = \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}, \qquad \mathbf{P} = \mathbf{D}^{-\frac{1}{2}}\mathbf{Q}\mathbf{D}^{\frac{1}{2}}. \tag{26}$$

Since $\mathbf{Q}$ is symmetric, it has the eigen decomposition of $\mathbf{Q} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$. In addition, we know that in practice all $\mathbf{Q}$'s eigenvalues are in (−1,1] and $\mathbf{u} = \mathbf{D}^{\frac{1}{2}}\mathbf{1}$ is its eigenvector for eigenvalue $\lambda = 1$ (Supplementary Note 1). We also know that $\mathbf{P}$ shares the same eigenvalues as $\mathbf{Q}$ and its right eigenvectors $\mathbf{\Psi}$ and left eigenvectors $\mathbf{\Phi}$ are

$$\mathbf{\Psi} = \mathbf{D}^{-\frac{1}{2}}\mathbf{U}, \quad \mathbf{\Phi} = \mathbf{D}^{\frac{1}{2}}\mathbf{U} \tag{27}$$

Next, to speed up the calculation, we approximate diffusion maps using only the top n diffusion components (Supplementary Note 1), where n is a user-specified parameter with default value $n = 100$. First, we calculate the top n eigenvalues and eigenvectors of $\mathbf{Q}$ using the Implicitly Restarted Lanczos Method[56] (via scipy.sparse.linalg.eigsh function). We also provide the alternative option to calculate the top n eigenvalues using the randomized SVD algorithm[54] (Supplementary Note 1). We order these n eigenvalues by magnitude:

$$1 = \lambda_0 > |\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_{n-1}|, \tag{28}$$

and define a family of approximated diffusion maps $\{\mathbf{\Psi}_t\}_{t \in \mathbb{N}}$ parameterized by timescale t:

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

$$\mathbf{\Psi}_t(\mathbf{x}_i) = \begin{pmatrix} \lambda_1^t \psi_1(i) \\ \lambda_2^t \psi_2(i) \\ \vdots \\ \lambda_{n-1}^t \psi_{n-1}(i) \end{pmatrix}. \quad (29)$$

Note that we use the right eigenvectors of the transition matrix **P** to construct diffusion maps. Using the eigenvectors of **P** is consistent with the original diffusion map paper[14] and recommended in the spectral clustering literature[41]. The DPT paper[15] constructs diffusion maps using eigenvectors of the symmetric matrix **Q** instead.

We next define approximated diffusion pseudotime maps $\{\mathbf{\Psi}'_t\}_{t \in \mathbb{N} \cup \{\infty\}}$ based on approximated diffusion maps:

$$\mathbf{\Psi}'_t(\mathbf{x}_i) = \sum_{t'=1}^{t} \mathbf{\Psi}_{t'}(\mathbf{x}_i) = \begin{pmatrix} \lambda_1 \frac{1-\lambda_1^t}{1-\lambda_1} \psi_1(i) \\ \lambda_2 \frac{1-\lambda_2^t}{1-\lambda_2} \psi_2(i) \\ \vdots \\ \lambda_{n-1} \frac{1-\lambda_{n-1}^t}{1-\lambda_{n-1}} \psi_{n-1}(i) \end{pmatrix}. \quad (30)$$

In particular, when $t = \infty$, we recover the DPT method (except it uses the eigenvectors of **Q**). We wish to pick a timescale t that smoothens out most of the noise but little signal. We select t based on the von Neumann entropy[40] of the graph induced by each timescale. For each t, its power matrix $\mathbf{P}^t$ induces a graph with the following Laplacian and density matrices (Supplementary Note 1):

$$\mathbf{L}(t) = \mathbf{I} - \mathbf{P}^t, \quad \rho(\mathbf{L}(t)) = \frac{\mathbf{L}(t)}{trace(\mathbf{L}(t))}. \quad (31)$$

We derive the von Neumann entropy $S(t)$ for the top n diffusion components from the density matrix $\rho(\mathbf{L}(t))$ (Supplementary Note 1):

$$S(t) = -\sum_{i=0}^{n-1} [\eta'(t)]_i \cdot \log[\eta'(t)]_i, \quad where \quad [\eta'(t)]_i = \frac{1-\lambda_i^t}{\sum_{j=0}^{n-1} 1-\lambda_j^t}. \quad (32)$$

$S(t)$ increases as t increases and reaches its maximum $\log(n-1)$ when $t \to \infty$. Because smaller eigenvalues of **P** (likely representing noise) decrease to 0 (and hence contribute a 1 to the entropy) much more rapidly than large eigenvalues (likely representing signal)[39], we expect to observe a high rate of increase in $S(t)$ initially when noise is smoothed out and then a low rate of increase when signal begins to be removed (Extended Data Figure 4b). We pick the timescale t as the knee point of this von Neumann entropy curve.

Author Manuscript
Author Manuscript
Author Manuscript
Author Manuscript

Let us denote $p_t = \begin{bmatrix} t \\ S(t) \end{bmatrix}$ as the point for timescale t in the curve. We find the kneepoint by scanning all ts (where t is an integer) between 1 and $maxt$, and pick the t that is furthest from $p_2 - p_1$, the segment connecting the two endpoints of the curve. $maxt$ is a user-provided parameter set to $maxt = 5000$ by default. We calculate the distance between point p_t and the segment $p_2 - p_1$ as follows:

$$dis = \frac{\|(p_t - p_1) \times (p_2 - p_1)\|}{\|p_2 - p_1\|}. \tag{33}$$

For the selected t, given a user-specified cell $\mathbf{r}$ as the root, we calculate the diffusion pseudotime distance from root to any other cell $\mathbf{x}$ as

$$D_t'(\mathbf{r}, \mathbf{x}) = \|\mathbf{\Psi}_t'(\mathbf{r}) - \mathbf{\Psi}_t'(\mathbf{x})\|. \tag{34}$$

We then normalize the diffusion pseudotime distance into [0,1] as the diffusion pseudotime.

Modularity-based community detection algorithms—We construct a weighted undirected graph $G = (V, E, w)$ from the affinity matrix $\mathbf{W}$. In the graph, vertex set V contains all cells, and an edge $(u, v) \in E$ if and only if $\mathbf{W}_{u,v} > 0$. The weight of the edge is calculated as:

$$w(u, v) = \left\lfloor \frac{\mathbf{W}_{u,v}}{\text{median}_{i<j}\mathbf{W}_{i,j}} \times 100 \right\rfloor / 100. \tag{35}$$

Community detection algorithms try to find a partition C of cells that maximizes the following modularity function[57]:

$$Q = \frac{1}{2m} \sum_{c \in C} \left(e_c - \gamma \frac{K_c^2}{2m} \right), \tag{36}$$

where each $c \in C$ consists of cells in that community, γ is the resolution parameter controlling the total number of communities, and

$$m = \sum_{(u,v) \in E} w(u, v), \qquad e_c = \sum_{u \in c, v \in c, (u,v) \in E} w(u, v), \qquad K_c = \sum_{u \in c} \sum_{(u,v) \in E} w(u, v). \tag{37}$$

Pegasus supports two modularity-based community detection algorithms: Louvain[16] and Leiden[17]. For both algorithms, Pegasus sets the resolution $\gamma = 1.3$ by default and reports each community as a separate cluster.

The Louvain algorithm[16] optimizes the modularity function Q in two phases: (1) in the move phase, each node is inspected and moved to the community that yields the largest increase in

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Q; (2) in the aggregation phase, each community aggregates into a new node to form an aggregated graph. The algorithm starts from the partition that each cell is its own community and repeats the two phases until there is no increase in Q. Pegasus uses the louvain-igraph implementation from Vincent Traag [https://github.com/vtraag/louvain-igraph]. Note that the latest release of louvain-igraph package (v0.6.1) contains a bug that prevents it from being reproducible even when the same random seed is used. Thus, Pegasus installs this package directly from the github master branch, in which the bug is fixed.

The Leiden algorithm[17] is a recent improvement over the Louvain algorithm and consists of three phases: (1) a move phase, which is similar to Louvain's; (2) a refinement phase, when each community found in (1) is examined and may be split into sub-communities; (3) an aggregation phase, when each sub-community from (2) is aggregated into a new node and assigned to an initial partition based on communities from (1). Pegasus uses the leidenalg implementation from Vincent Traag [https://github.com/vtraag/leidenalg]. Applying the Leiden algorithm on communities detected from previous Leiden runs can further improve the modularity function[17]. Thus, following SCANPY[27], Pegasus runs the Leiden algorithm iteratively on the graph G until Q does not further improve (n_iterations = −1 by default).

Spectral-community-detection algorithms for fast clustering—Pegasus provides two spectral-community-detection algorithms: spectral-Louvain and spectral-Leiden. Spectral-community-detection algorithms aggregate cells into thousands of groups of cells, where each group consists of cells that are likely from the same "real" cluster, and then apply community detection algorithms such as Louvain and Leiden on the groups instead of on individual cells to achieve a major speedup.

Our variant of the spectral clustering[41] algorithm partitions cells into groups by applying the k-means algorithm on calculated diffusion pseudotime component space, using a 2-level clustering strategy. We first partition the cells into k_1 clusters using the scikit-learn's KMeans function with default parameters. We further partition each of the k_1 clusters into k_2 sub-clusters using the KMeans function with $n_init = 1$. This procedure will give us $k_1 \cdot k_2$ cell groups, on which we then apply community detection algorithms. Pegasus sets $k_1 = 30$ and $k_2 = 50$ by default. If diffusion pseudotime components are not calculated, Pegasus will apply the k-means algorithm on the PC space instead.

t-SNE, UMAP and FLE—Pegasus calculates a t-SNE using the Multicore-TSNE package implemented by Dmitry Ulyanov [https://github.com/DmitryUlyanov/Multicore-TSNE]. We found and fixed a random-seed-related bug in this package that prevents the package from reproducing the exact t-SNE coordinates, and provide a bug-free version of the package at [https://github.com/lilab-bcb/Multicore-TSNE]. Pegasus uses the following t-SNE parameters by default: perplexity=30, early_exaggeration=12, learning_rate=1000, n_iter=1000 and n_iter_early_exag=250.

Pegasus calculates a FIt-SNE, which are fast approximations of t-SNE embeddings, using the pyFIt-SNE package from Kluger lab [https://github.com/KlugerLab/pyFIt-SNE]. Pegasus uses the following FIt-SNE parameters by default: perplexity=30,

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

early_exaggeration=12, learning_rate=1000, max_iter=1000, stop_early_exag_iter=250 and mom_switch_iter=250.

Pegasus calculates a UMAP based on the k-NN graph constructed by the HSNW algorithm, using the umap package from Leland McInnes [https://github.com/lmcinnes/umap]. Pegasus uses the following UMAP parameters by default: n_neighbors=15, min_dist=0.5, spread=1.0, n_epochs=250 and learning_rate=1.0.

Pegasus calculates an FLE using the forceatlas2 package, a Java package developed for scSVA[24]. The forceatlas2 package improved upon the Gephi's ForceAtlas2[22] Java code by providing more efficient parallelization[24] and stops iterations if the average distance between FLE coordinates in adjacent iterations is no greater than δ or the maximum number of iterations n_δ is reached, where δ and n_δ are user-specified parameters. Pegasus sets $\delta = 2.0$ and $n_\delta = 5000$ by default.

Deep-learning-based visualization—To generate Net-* embeddings, we first subsample cells by local density. To estimate a proxy of the local density for each cell we denote as d_i the distance from cell i to its kth nearest neighbor in the calculated k-NN graph, where k is a user-provided parameter set to k = 25 by default. d_i is inversely proportional to the local cell density such that p_i, the probability of sampling cell i, is

$$p_i \propto \frac{1}{d_i^{\alpha}}, \tag{38}$$

where α is a parameter that determines how much local density should influence the sampling process. If $\alpha = 0$, we recover uniform sampling. In Pegasus, we set $\alpha = 1$. We then subsample P percent of cells based on sampling probabilities $\{p_i\}$, where P is a user-specified parameter set to 10% by default.

Next, we train a deep-learning-based regressor, using a neural network with 1 input layer and 4 hidden layers. The input layer connects with the top m principal components for each cell and the 4 hidden layers contain 100, 75, 50, and 25 ReLU units respectively. We train the network using scikit-learn's MLPRegressor with a stochastic gradient decent solver, an adaptive learning rate and a $L2$ penalty parameter of 0.1. We additionally scale the inputs and outputs of the neural network, such that the maximal standard deviation of the inputs and outputs are 1.0 and 15.0 respectively.

In the final refinement step, Pegasus changes the following parameters for each embedding method. For t-SNE: learning_rate= $0.33 \times N$, where N is the total number of cells, n_iter=150 and n_iter_early_exag=0. For UMAP: learning_rate=10 and n_epochs=30. For FLE: $n_\delta = 1500$.

Differential expression analysis—Pegasus can perform Welch's t-test, Fisher's exact test and the Mann-Whitney U test between cells within and outside of a cluster. It controls the False Discovery Rate (FDR) at 5% using the Benjamini-Hochberg procedure[58]. Pegasus can optionally calculate the Area Under the ROC curve (AUROC) for each gene by considering the binary classification problem that uses the gene as the only feature to predict

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

if a cell is within or outside of a cluster. Pegasus performs these analyses in parallel across genes to speed up the calculation process and outputs results for all genes in a spreadsheets. If AUROC is calculated, genes are ranked by their AUROC values.

Feature importance scores from LightGBM—Pegasus trains a LightGBM[25] classifier on the log expression matrix to predict cluster labels. It uses 90% of the cells as the training data and 10% of the cells as the test data. It stops training the classifier once the prediction accuracy on the test dataset drops, and then extract each gene's feature importance score from the trained classifier. To assign genes with high importance scores to each cluster, for each gene with a high importance score, it clusters the mean log expression levels of all clusters into 3 groups using the *k*-means algorithm, and assigns each of the 3 groups as strongly up-regulated, weakly up-regulated and down-regulated based on the mean log expression of the group. It then removes the group with largest size (as not significant) and assigns the gene to clusters in the other two groups.

Marker-based cell type annotation—To annotate cell types, Pegasus first loads known marker genes for each cell type from a user-provided JSON file. Each marker gene is associated with a sign (positive or negative) and a weight. The format of the JSON file is defined at [https://cumulus-doc.readthedocs.io/en/latest/cumulus.html#how-cell-type-annotation-works]. For each cluster, Pegasus enumerates all putative cell types and calculates a score between 0 and 1 per cell type, describing how likely cells from the cluster are to be of the specific cell type. To calculate the score, Pegasus assigns each marker a maximum impact value of 2. For a positive marker, if it is not up-regulated, its impact value is 0. Otherwise, if *fc* the fold change in the percentage of cells expressing this marker (within *vs.* outside of the cluster) satisfies *fc* 1.5, it has an impact value of 2 and is recorded as a strong supporting marker. If $fc < 1.5$, it has an impact value of $1 + \frac{fc - 1}{0.5}$ and is recorded as a weak supporting marker. For a negative marker, if it is up-regulated, its impact value is 0. If it is neither up-regulated nor down-regulated, its impact value is 1. Otherwise, if $\frac{1}{fc} \geq 1.5$, it has an impact value of 2 and is recorded as a strong supporting marker. If $\frac{1}{fc} < 1.5$, it has an impact value of $1 + \frac{1/fc - 1}{0.5}$ and is recorded as a weak supporting marker. The overall score is calculated as the ratio between the sum of impact values and the sum of weights multiplied by 2 from all expressed markers. Pegasus will evaluate all possible cell subtypes recursively if the current cell type score is no less than 0.5. Finally, Pegasus reports putative cell types with scores no less than *minimum_report_score* in descending order with respect to cell type scores for all clusters. By default, Pegasus sets *minimum_report_score* = 0.5.

BENCHMARKING EXPERIMENTS

Benchmarked tools and benchmarking environments—We tested each component of Pegasus v0.15.0 and benchmarked it with SCANPY v1.4.4.post1 and Seurat v3.1.0 on key analysis tasks (Supplementary Tables 2 and 3) using a high-performance local server (Cloud-based times as part of a Cumulus workflow are in Table 1). The server has 28 CPU threads (1 Intel Xeon E5–2660v4 processor; 14-Core 2.00GHz, 35MB Cache) and 256 GB

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

DDR4 ECC registered memory. In benchmarking, we used all 28 CPU threads whenever possible. For reproducibility, we have prepared a Docker image that has all three tools and their dependencies installed. This Docker image also contains instructions on how to reproduce the results shown in this manuscript. The Docker image is available at [https://hub.docker.com/r/cumulusprod/cumulus-experiment].

We also benchmarked Cumulus on cloud with running Cell Ranger + Seurat/SCANPY pipeline (on a 32 CPU-thread, 120GB Google Cloud virtual machine) on the bone marrow dataset. We used Cell Ranger v2.2.0 for this benchmark. The virtual machine for running Seurat (v3.1.0) and SCANPY (v1.4.4.post1) is created on Google compute engine zone us-west1-d, the same zone on which Cumulus ran its analysis step.

Bone marrow datasets pre-processing—We preprocessed the bone marrow data set by filtering out any cell with fewer than 500 genes or more than 6,000 genes, or where at least 10% of UMIs were from mitochondrial genes, retaining 274,182 cells. We then selected robust genes with $x = 0.05\%$, normalized expression into TP100K and log-transformed the expression matrix.

HVG selection—We applied the standard and new HVG selection procedures to the log-transformed expression matrix separately, followed by the same downstream analyses on the resulting two sets of HVGs, using Pegasus with default parameters: batch correction, dimensionality reduction via PCA, k-NN graph construction, community detection using the Louvain algorithm, 2D visualization using FIt-SNE, differential expression analysis and marker-based cell type annotation.

We compared the highly variable genes selected using the two procedures with a list of immune genes curated by the ImmPort[28] team (Extended Data Figure 1b) from [https://www.immport.org/shared/geneData/GOappend1.xls], which contains 1,534 genes (with duplicates) annotated with immune-related gene ontology (GO) terms. The comparison results are available in Supplementary Data 1.

We evaluated the similarity between clusters obtained using the two HVG selection procedures with the adjusted mutual information[59] (AMI) score defined below:

$$AMI(U, V) = \frac{I(U, V) - E\{I(U, V)\}}{\frac{1}{2}[H(U) + H(V)] - E\{I(U, V)\}}, \quad (39)$$

where U and V represent two cluster settings, H denotes entropy and I denotes mutual information.

Benchmarking of batch correction methods—We benchmarked Pegasus, ComBat, MNN, BBKNN and Seurat V3 using a subset of data from the bone marrow dataset, consisting of the first 10x Genomics channel from each of the 8 donors. Following the preprocessing steps above, we retained 34,654 cells. We then applied the new HVG selection procedure and downstream analyses described above (without batch correction) to obtain cell-type-annotated clusters (Extended Data Figure 2b).

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

The clustering results showed one particular donor-specific effect (Extended Data Figure 2b), with one donor-3-specific CD14^{+} monocyte cluster and one donor-3-specific T cell cluster. Since we do not want to count this donor-specific effect as "biology" when we compute kSIM acceptance rates, we constructed a ground truth for kSIM acceptance rates as follows. First, we merged the monocyte cluster into the larger monocyte cluster to its right. The donor-3-specific T cell cluster is adjacent to five other T cell clusters. We trained a LightGBM[25] classifier that predicted cluster labels based on log expression levels using cells from the 5 clusters (90% training data + 10% validation data). The classifier's accuracy on test data was 85.4%. We then used this classifier to assign each cell in the donor-3-specific T cell cluster into one of the 5 adjacent T cell clusters.

To generate batch-corrected results, we applied the same preprocessing step to the 34,654 cell dataset, selected top 2,000 HVGs using the new batch-aware HVG selection procedure, and extracted the HVG-specific gene-count matrix. With Pegasus, we applied the L/S adjustment method to the matrix to obtain batch-corrected expression levels. With SCANPY, we obtained ComBat-corrected and MNN-corrected expression levels. With Seurat v3 we obtained Seurat-corrected expression levels. We performed PCA, k-NN graph construction, and UMAP on the corrected expression matrices. With BBKNN, we used its output k-NN graph to replace Pegasus's k-NN graph and kept other analyses the same.

We used kBET and kSIM acceptance rates on UMAP 2D coordinates. For kBET acceptance rate, we set $k = 25$ and $\alpha = 0.05$. For kSIM acceptance rate, we set $k = 25$ and $\beta = 0.9$.

Benchmarking approximate nearest neighbor finding methods—We benchmarked the approximate nearest neighbor finding algorithms used by Pegasus, SCANPY and Seurat on the bone marrow dataset with default parameters. Pegasus ran the HNSW algorithm in full speed mode, SCANPY used the algorithm implemented in UMAP, and Seurat used the RcppAnnoy package at [https://cran.rstudio.com/web/packages/RcppAnnoy/index.html]. We ran the three methods on coordinates from the top 50 PCs produced by Pegasus and sought for top 100 nearest neighbors (including the cell itself). We also ran the brute force k-NN searching algorithm using scikit-learn[55] to compute the ground truth. We evaluated each method's performance using recall, defined as the percentage of k nearest neighbors that are also in the ground truth, and speed.

Diffusion pseudotime maps—We preprocessed the bone marrow dataset, selected HVGs using the new procedure, corrected batch effects, ran PCA and k-NN graph construction as described in HVG selection section. We generated diffusion pseudotime maps (with the parameters noted in Figure 2b and Extended Data Figure 4), visualized diffusion pseudotime maps using the FLE algorithm, and annotated the resulting trajectories using the cell type annotation described in the HVG selection section (with new HVG procedure).

Spectral community detection algorithms—We preprocessed the bone marrow dataset as described in HVG selection section, selected HVGs, corrected batch effects, ran PCA, k-NN graph construction, calculated a diffusion pseudotime map and performed 2D visualization using FIt-SNE with default parameters. We then generated different cluster

Author Manuscript

settings using either the spectral clustering, Louvain, spectral-Louvain, Leiden, or spectral-Leiden algorithm. We performed differential expression analysis and marker-based cell type annotation for each of the clusterings separately.

Deep-learning-based visualization—We preprocessed the bone marrow dataset as described in HVG selection section, selected HVGs, corrected batch effects, ran PCA, k-NN graph construction, diffusion pseudotime map calculation using default parameters. We then ran Net-tSNE vs. t-SNE, Net-UMAP vs. UMAP and Net-FLE vs. FLE and annotated each visualization using cell types calculated in the HVG selection section (with the new HVG procedure).

Benchmarking Pegasus, SCANPY and Seurat on the full bone marrow dataset —We benchmarked Pegasus, SCANPY and Seurat on 10 tasks using the full bone marrow data of 274,182 cells. To ensure a fair comparison, whenever possible, all three methods received the same input computed using Pegasus with default parameters for each task. The only exception is Seurat, which does not accept a pre-computed affinity matrix as input for clustering. Thus, we provided Seurat with pre-computed principal components (PCs) instead and asked it to compute the affinity matrix before clustering. In addition, we used the *future* [https://cran.r-project.org/web/packages/future/] framework for parallelization as suggested at https://satijalab.org/seurat/v3.0/future_vignette.html. For the batch correction step, we made each 10x channel as a separate batch, which resulted in 63 batches in total. We used Seurat v3's integration method[33], BBKNN[32] and L/S adjustment method[29] to correct batch effects in Seurat, SCANPY and Pegasus, respectively. For *k*-NN graph construction, we set k = 100 for all three methods and set Pegasus in full speed mode. For Louvain-like and Leiden-like clustering, Pegasus used spectral-Louvain and spectral-Leiden algorithms, and Seurat and SCANPY used Louvain and Leiden algorithms. For t-SNE-like visualization, Seurat and Pegasus used FIt-SNE and SCANPY used Multicore-TSNE. For UMAP-like visualization, Pegasus used Net-UMAP, and Seurat and SCANPY used UMAP. We excluded the *k*-NN graph construction times for SCANPY and Pegasus, because it was accounted for in the *k*-NN graph construction task. Seurat calculates the *k*-NN graph again in the UMAP step using the *umap* Python package and thus we included *k*-NN graph construction time. For FLE-like visualization, Pegasus used Net-FLE and SCANPY used the fa2 package [https://github.com/bhargavchippada/forceatlas2]. The function calls, commands and parameters used for the three tools can be found in Supplementary Note 2.

Benchmarking Pegasus, SCANPY and Seurat v3 on the 1.3 million mouse brain dataset—We obtained the 1.3 million mouse brain dataset from https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.3.0/1M_neurons. The unfiltered data contains 1,306,127 cell barcodes. We preprocessed this data set by filtering out any cell with fewer than 500 genes or more than 6,000 genes, or with at least 10% of UMIs from mitochondrial genes. After filtration, we retained 1,286,072 cells. We selected robust genes with $x = 0.05\%$, normalized expression into TP100K and log-transformed the expression matrix. Next, we benchmarked the three methods as described above. Since the fa2 package was too slow for our dataset, instead of running it for 5,000 iterations, we only ran it for 500 iterations and estimated total time by multiplying a factor of 10. The function

calls, commands and parameters used for the three tools can be found in Supplementary Note 2.

Benchmarking Pegasus components by subsampling the bone marrow dataset—We uniformly subsampled the bone marrow dataset to 5K, 10K, 25K, 50K, 100K and 200K cells. Together with the full dataset, this resulted in 7 datasets of various sizes. We then benchmarked Pegasus on the 7 datasets and recorded the execution time of each Pegasus components, including highly variable gene selection (hvg), batch correction, PCA, KNN graph construction (knn), diffusion pseudotime map (diffmap), Louvain and spectral-Louvain algorithms, Leiden and spectral-Leiden algorithms, multicore t-SNE (tsne), FIt-SNE, UMAP, Net-UMAP, FLE and Net-FLE. We benchmarked Pegasus using two settings – cloud setting (28 threads) and laptop setting (8 threads) – to test if it works efficiently in both cloud and personal laptop environments. The benchmarking results are recorded in Supplementary Data 2. FIt-SNE, Net-UMAP, Net-FLE, spectral-Louvain and spectral-Leiden are improvements over multicore t-SNE, UMAP, FLE, Louvain and Leiden. Supplementary Data 2 suggests these improvements are faster in both cloud and laptop settings once the number of cells reaches 25K. In addition, these improvements are as efficient as their counterparts for small datasets (i.e. 5K and 10K). The Pegasus commands can be found in Supplementary Note 2.

Benchmarking Pegasus, SCANPY and Seurat v3 on the 5K PBMC dataset—We obtained the filtered gene-count matrix of the 5K PBMC dataset from http://cf.10xgenomics.com/samples/cell-exp/3.0.2/5k_pbmc_v3/5k_pbmc_v3_filtered_feature_bc_matrix.h5. This matrix contains 5,025 cell barcodes. We preprocessed this data set by further filtering out any cell with fewer than 500 genes or more than 6,000 genes, or with at least 20% of UMIs from mitochondrial genes. After filtration, we retained 4,515 cells. We selected robust genes with $x = 0.05\%$, normalized expression into TP100K and log-transformed the expression matrix. Next, we benchmarked the three methods as described in the full bone marrow dataset, except that we did not benchmark batch correction since this dataset contains only one batch. The function calls, commands and parameters used for the three tools can be found in Supplementary Note 2.

Measuring Pegasus, SCANPY and Seurat v3 peak memory on the 5K PBMC and bone marrow datasets—We measured the peak memory usage of Pegasus, SCANPY and Seurat v3 on the 5K PBMC and bone marrow datasets. To obtain realistic peak memory measurements, we asked each tool to process the same dataset from the beginning, instead of feeding the same input for all three tools per task and aimed to use identical or similar parameters by all three tools. In addition, for all three tools, we did not execute the tasks that Seurat does not provide in Supplementary Table 4 to make sure the measured peak memory usages are comparable. We used 8 threads for the 5K PBMC dataset and 28 threads for the bone marrow dataset. The function calls, commands and parameters used for the three tools can be found in Supplementary Note 2.

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Cloud computing execution time and cost—Cumulus utilizes Google Cloud Platform's preemptible instances. Jobs running in preemptible instances can be kicked off by others' jobs with higher priority but are 5x cheaper. By default, Cumulus allows up to 2 tries using preemptible instances before switching to non-kicked-off instances. Cumulus execution time is read out from Terra execution logs. To estimate the execution time of *mkfastq* and *count* steps on a 32 CPU-threads virtual machine (Table 1), we only sum over Terra-reported Docker running times of successful *mkfastq* and *count* runs, respectively. The *analysis* time for Seurat and SCANPY are estimated by running each tool on a same 32 CPU threads, 120 GB memory Google Cloud virtual machine instance. Since both Seurat and SCANPY do not have functions to aggregate 10x samples into a big count matrix, we used the aggregated matrix produced by Cumulus as their input and excluded the matrix aggregation time. In addition, since Seurat's batch correction failed in the previous benchmarking (Figure 2e), we reduced the number of batches from 63 to 8 (one per donor) and the number of PCs used from 30 (default) to 20. See Supplementary Note 2 for details on how Seurat, SCANPY and Cumulus analysis step were run. The total computational costs were reported by Terra and we calculated the average cost per sample by dividing 63.

Scalability analysis of the *count* step—The full bone marrow dataset consists of 63 channels and Cumulus ran them in parallel across separate computing nodes in the *count* step. We collected the execution time of each channel from Terra and compiled them into Supplementary Data 3. Since Cumulus allowed 2 preemptible tries for each *count* job, the execution time includes the time of both the failed preemptible tries and the final successful run. In addition, we calculated the amortized cost per channel (= total cost / total runtime * one count job runtime) in Supplementary Data 3. We plotted execution time and amortized total cost against channels in Extended Data Figure 7.

Gene-count matrices produced on the cloud and the local server are identical —Cumulus workflows are written in WDL and executed by Cromwell [https://cromwell.readthedocs.io/en/stable/], which has both cloud and local backends. We ran Cumulus on both cloud and a local server (96 CPU threads, 1 TB memory) to generate gene-count matrices for one 10x Genomics V3 dataset and one SMART-seq2 dataset and confirmed the matrices were identical. Specifically, we chose a 5K peripheral blood mononuclear cells (PBMC) dataset of a healthy donor from 10x Genomics website [https://support.10xgenomics.com/single-cell-gene-expression/datasets/3.0.2/5k_pbmc_v3] and ran Cell Ranger v3.1.0 with Cell Ranger human reference GRCh38 v3.0.0. For SMART-seq2, we picked 4 single-end 50 base pair (bp) and 4 paired-end 100 bp mouse embryonic stem cells sequenced by BGISEQ-500[60] and ran RSEM+HISAT2 with a transcript reference extracted from mouse genome GRCm38 using a filtered Ensembl 93 mouse gene annotation file. Both cloud and local-server generated count matrices are available in Supplementary Data 4.

Benchmarking Cell Ranger, Optimus, Alevin, Kallisto-BUStools and STARsolo —We downloaded the FASTQ files of the human 5K PBMC dataset from the 10x Genomics website [https://support.10xgenomics.com/single-cell-gene-expression/datasets/3.0.2/5k_pbmc_v3] as the test data and merged FASTQs from different sequencing lanes

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

into one FASTQ file for Optimus, Alevin, Kallisto-BUStools and STARsolo. We then asked Cumulus to run Cell Ranger (v3.1.0), Optimus (v1.4.0+, forked from GitHub master branch), Alevin (v1.1), Kallisto-BUStools (v0.24.4) and STARsolo (v2.7.3a) on the cloud with default parameters. We set preemptible as 2 for Cell Ranger, Alevin, Kallisto-BUStools and STARsolo and used default preemptible settings for Optimus. We summarized the benchmarking results in Supplementary Table 8. For running Optimus, our implementation includes two extra steps that are not part of the Optimus workflow: convert reference to be Optimus-compatible and copy Optimus outputs to a dedicated Google bucket location. We deducted the execution time of these two steps from Optimus' total execution time.

Life Sciences Reporting Summary—Further information related to statistics, data, software and study design is available in the Nature Research Reporting Summary linked to this article.

Data availability—The bone marrow dataset is available at https://data.humancellatlas.org/explore/projects/cc95ff89–2e68–4a08-a234–480eca21ce79. The 1.3 million mouse brain data set is available at https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.3.0/1M_neurons. The 5K PBMC data is available at https://support.10xgenomics.com/single-cell-gene-expression/datasets/3.0.2/5k_pbmc_v3. The BGISEQ SMART-Seq2 data is available at https://www.ncbi.nlm.nih.gov/sra?linkname=bioproject_sra_all&from_uid=430491. In particular, data from accessions SRX3654625, SRX3654622, SRX3654623, SRX3654630, SRX3654942, SRX3654606, SRX3654816 and SRX3654921 are used.

Code availability—Cumulus code consists of four components: the Pegasus and scPlot python packages, the Cumulus WDL workflows and Dockerfiles, the Cumulus docker images and the Cirrocumulus app. Pegasus source code is available at https://github.com/klarman-cell-observatory/pegasus. Pegasus documentation is available at https://pegasus.readthedocs.io. scPlot source code is available at https://github.com/klarman-cell-observatory/scPlot. We wrote all workflows using the Workflow Description Language (WDL, https://github.com/openwdl/wdl) and encapsulated all software packages into Docker images using Dockerfiles. Cumulus WDL and Dockerfiles are available at https://github.com/klarman-cell-observatory/cumulus. The source code used to generate feature-count matrices, 'generate_count_matrix_ADTs', are available at https://github.com/klarman-cell-observatory/cumulus_feature_barcoding. Cumulus Docker images are available at https://hub.docker.com/u/cumulusprod. For Terra users, we additionally deposit Cumulus workflows in the Broad Methods Repository https://portal.firecloud.org/?return=terra#methods and provide a step-by-step manual at https://cumulus.readthedocs.io. Cirrocumulus source code is available at https://github.com/klarman-cell-observatory/cirrocumulus. Cirrocumulus documentation is available at https://cirrocumulus.readthedocs.io. Pegasus, scPlot, Cumulus WDL files and Docker files, and Cirrocumulus are licensed under a BSD 3-clause license. In addition, we documented licenses for Cumulus dependencies in Supplementary Data 5. Due to 3rd party licensing requirements, we can only provide CellRanger dockers without bcl2fastq2 and users can

build their private bcl2fastq2-containing Dockers by following instructions listed in Cumulus documentation.

Extended Data

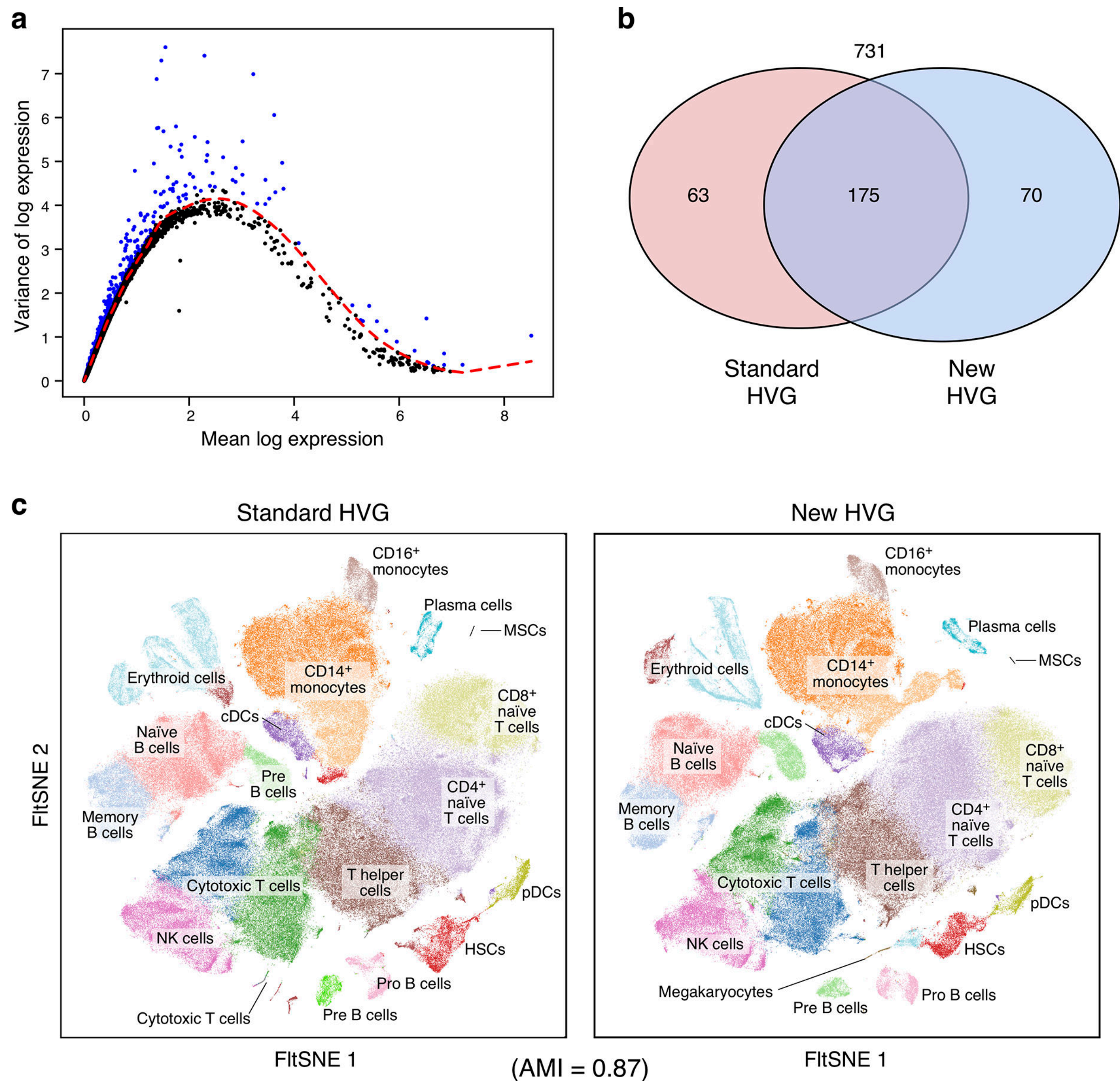
Author Manuscript

Author Manuscript

Author Manuscript

Author Manuscript

Author Manuscript Author Manuscript Author Manuscript Author Manuscript



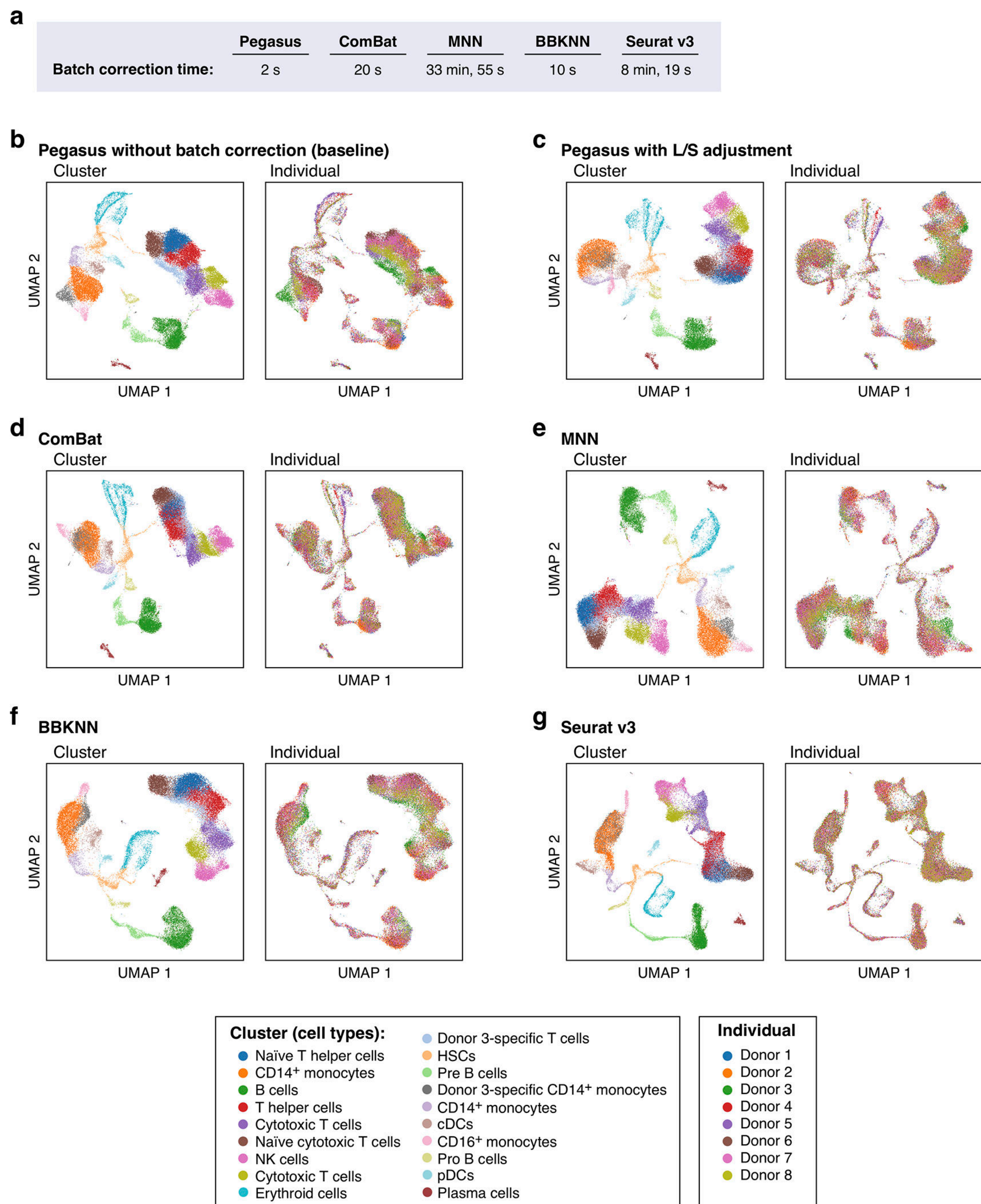
Extended Data Fig. 1. The new HVG selection procedure provides excellent quality vs. the standard procedure.

a. New HVG selection procedure (n = 16,613 robust genes). Variance (y axis) vs. mean (x axis) of log expression. Red: fit LOESS curve. HVGs (blue) are defined as the genes above the LOESS curve. **b.** Curated immune genes captured by each procedure. The number of ImmPort curated immune genes selected by a standard HVG procedure (red) and Cumulus (blue). **c.** Analysis with HVGs by new approach highlighted an additional cell type (n = 274,182 bone marrow cells). FIt-SNE plots of cells from the bone marrow dataset generated by Cumulus with HVG genes selected by the standard (left) and new (right) procedure and colored by cell subset annotations. Bottom: Adjusted Mutual Information (AMI) score shows overall high concordance. Only the plot from the new procedure identifies megakaryocytes. HSCs: hematopoietic stem cells; MSCs: mesenchymal stem cells; cDCs:

conventional dendritic cells; pDCs: plasmacytoid dendritic cells; NK cells: natural killer cells.

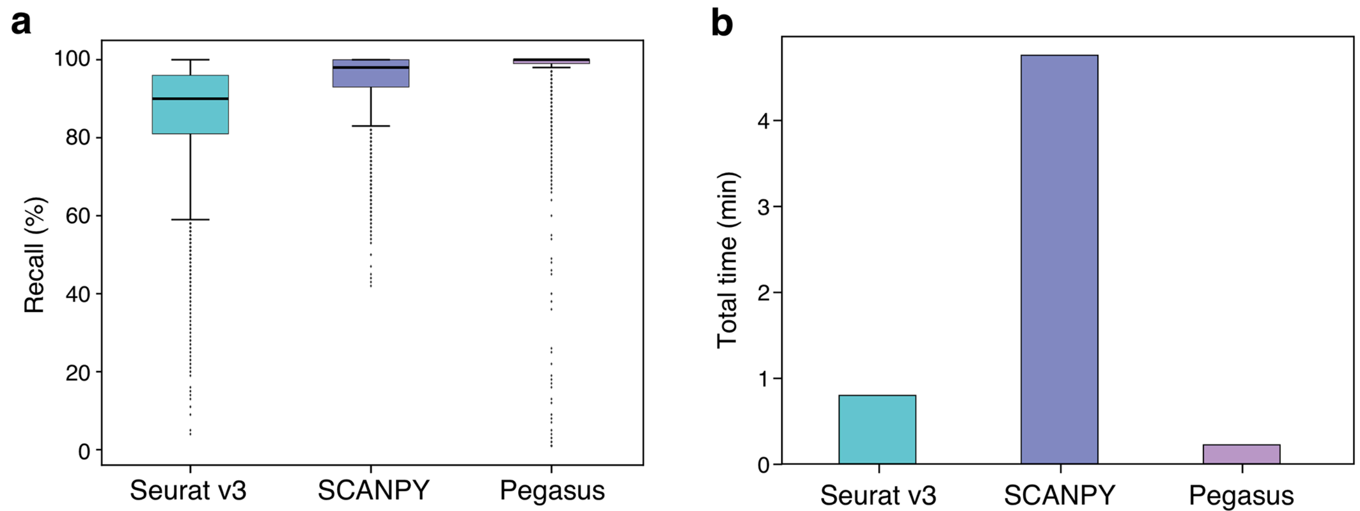
Author Manuscript
Author Manuscript
Author Manuscript
Author Manuscript

Author Manuscript
Author Manuscript
Author Manuscript
Author Manuscript



Extended Data Fig. 2. Benchmarking of batch correction methods on 34,654 bone marrow cells.
a. Execution time of each method. **b.-g**. UMAP visualizations of the bone marrow cells (n = 34,654) colored by either cell type annotation (left) or donor identity (right) without batch correction (**b**, baseline), Pegasus with L/S adjustment (**c**), ComBat (**d**), MNN (**e**), BBKNN (**f**), and Seurat v3 (**g**).

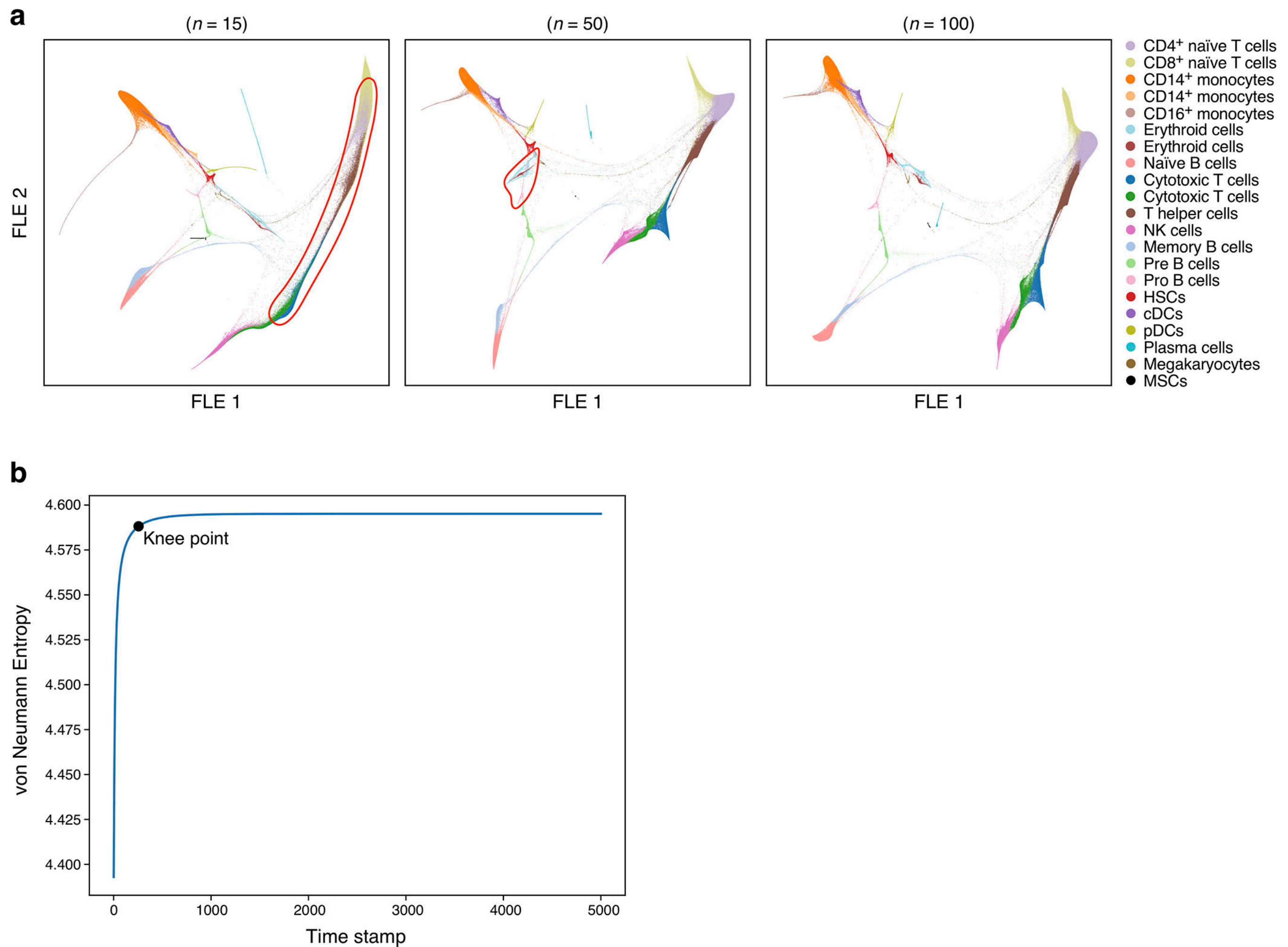
Author Manuscript
Author Manuscript
Author Manuscript
Author Manuscript



Extended Data Fig. 3. Benchmarking of approximate nearest neighbor finding methods on the bone marrow dataset (n = 274,182 cells).

Accuracy (**a**, y axis, % recall, Methods) and speed (**b**, y axis, minutes) of each of three methods. Boxplot (a): Line: median; box boundaries: lower and upper quartiles; whiskers: 1.5 interquartile range (IQR) below and above the low and high quartile, respectively.

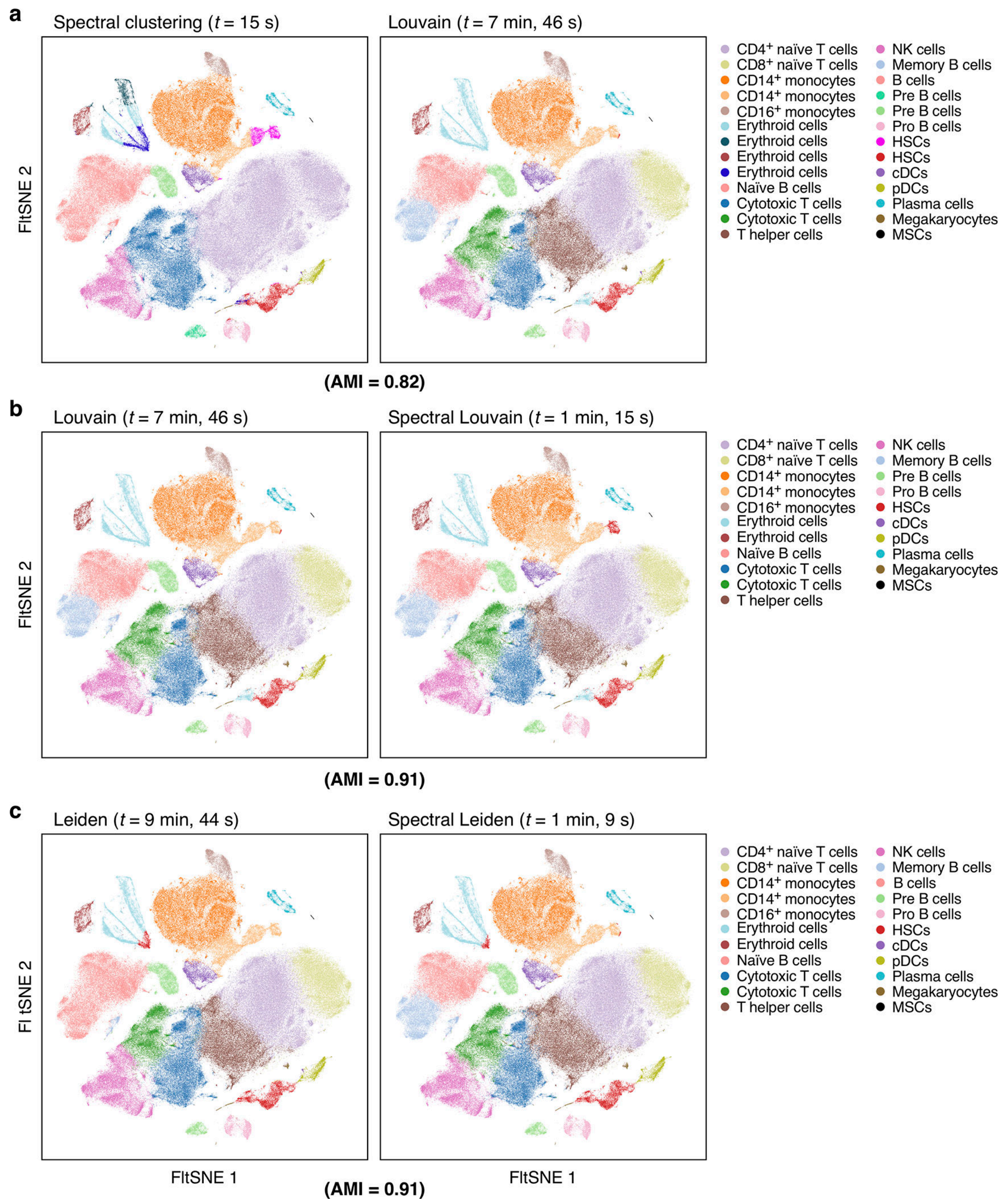
Author Manuscript Author Manuscript Author Manuscript Author Manuscript



Extended Data Fig. 4. Adjusting diffusion pseudotime map parameters for visualization of pseudotemporal trajectories.

a. Using a large number of diffusion pseudotime components yields a developmental trajectory that enhances separation of trajectories of different cell populations (n = 274,182 bone marrow cells). FLEs of single cell (colored by cell type annotation) generated from diffusion pseudotime maps ($t = \infty$) with 15 (left), 50 (middle) or 100 (right) components. CD8+ and CD4+ naïve T cells are fused together in the left FLE (circled in red). Erythrocytes and Pro-B cells are overlapped in the middle FLE (circled in Red). **b**. Choosing the timescale t for a diffusion pseudotime map. Von Neumann entropy (y axis) for diffusion maps with 100 components calculated from the bone marrow data at different timescales (x axis). Black point: knee point.

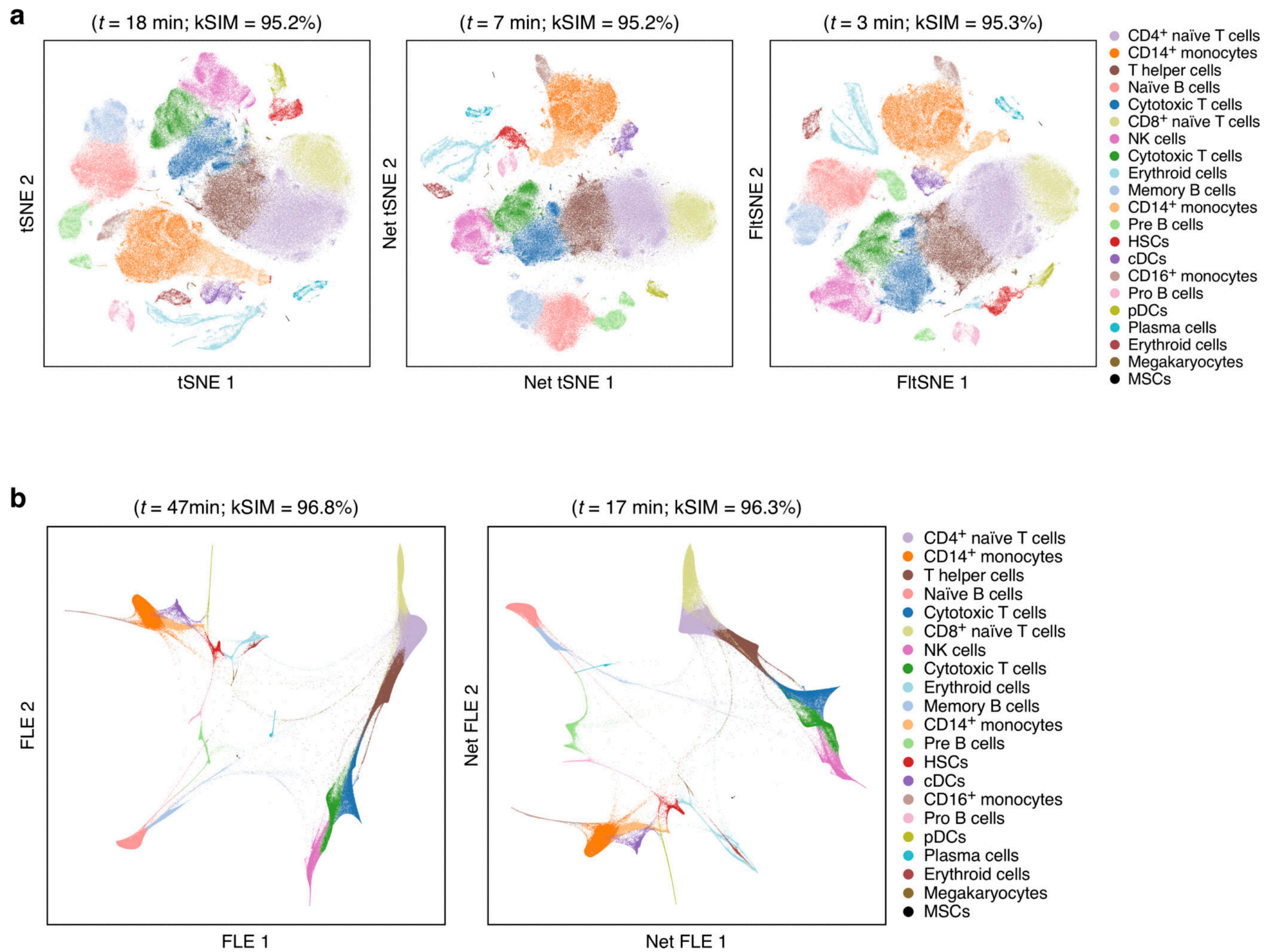
Author Manuscript Author Manuscript Author Manuscript Author Manuscript



Extended Data Fig. 5. Spectral community detection algorithms combine the strengths of spectral clustering and community detection algorithms.

FIt-SNE of bone marrow single cells (dots, n = 274,182) colored by cluster assignment from (**a**) Spectral (left) vs. Louvain (right) clustering; (**b**) Louvain (left) vs. Spectral Louvain (right) clustering; or (**c**) Leiden (left) vs. spectral Leiden (right) clustering. Top: Execution time; bottom: Adjusted Mutual Information (AMI). Post hoc annotation labels are listed.

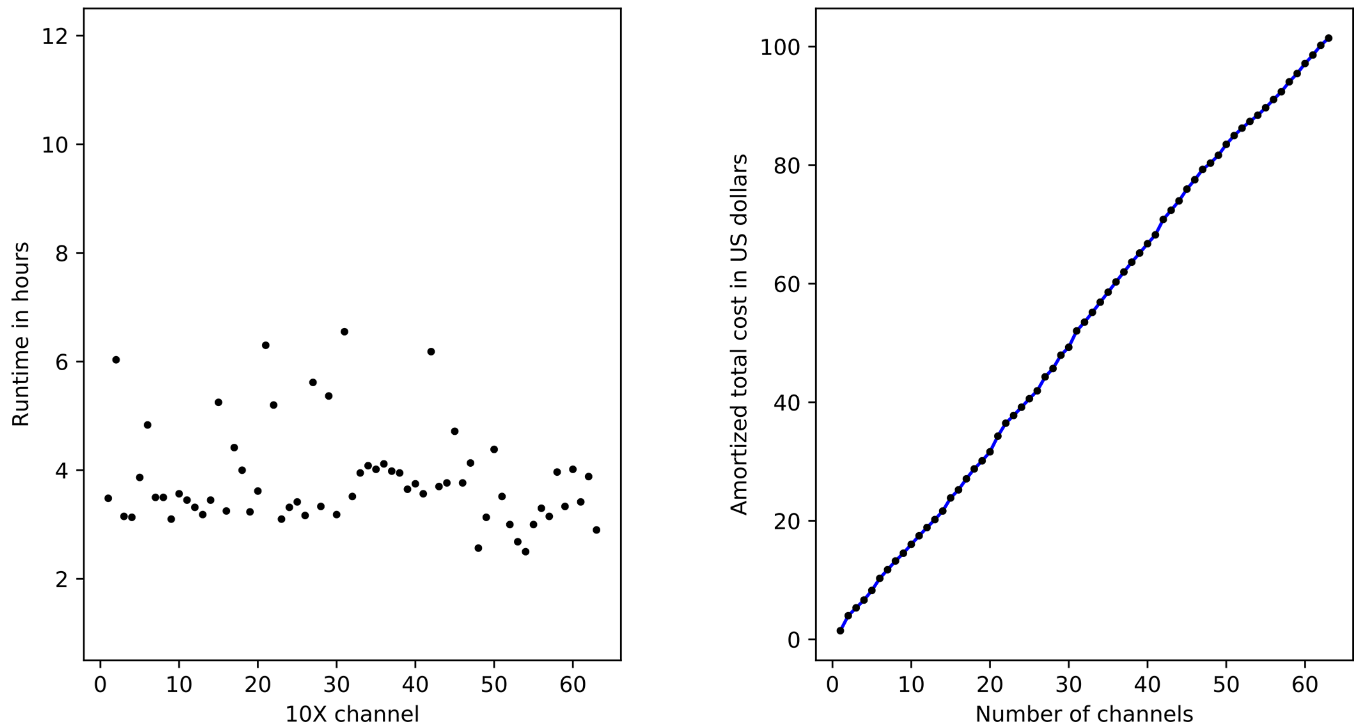
Author Manuscript



Extended Data Fig. 6. Deep-learning based visualization speeds up t-SNE and FLE visualizations while maintaining comparable quality.

Visualization of cell profiles (dots) from the full bone marrow data set (n = 274,182) colored by the same Louvain cluster membership (color; legend shows post hoc annotations) and laid out by (**a**) t-SNE (left), Net-tSNE (middle), or FIt-SNE (right); or (**b**) by FLE (left) or Net-FLE (right). Top: Execution time and kSIM acceptance rate.

Author Manuscript Author Manuscript Author Manuscript Author Manuscript



Extended Data Fig. 7. Benchmark the count step with respect to number of channels for the bone marrow dataset.

Plot of maximum runtime in hours (left) and amortized total costs (right) in US dollars against the number of 10x channels.

Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

Acknowledgments

We thank J. Rood for help with manuscript editing, L. Gaffney for help with figure preparation, E. Banks and A. Philippakis for advice on creating Cumulus' featured Terra workspace, C. O'Day and E. Law for advice on licensing Cumulus as open source, and C. O'Day additionally for help on summarizing the license terms of third-party packages on which Cumulus depends (Supplementary Data 5), D. Dionne, J. Waldman, J. Lee and K. Shekhar for their contribution in generating the preview data set and sharing it openly pre-publication, M. Maarouf and D. Erdogan for transferring the Pegasus namespace on Read the Docs to us, and J. Gatter for providing Kallisto-BUStools docker and WDLs. This work was supported by the Klarman Cell Observatory, the Manton Foundation, HHMI and the Ludwig Center at MIT (to A.R.) and the HTAPP scientific team at Leidos Biomedical Research, Inc., Frederick National Laboratory for Cancer Research and NCI.

References

1. Regev A et al. The human cell atlas white paper. arXiv:1810.05192 [q-bio] (2018).
2. Macosko EZ et al. Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets. Cell 161, 1202–1214 (2015). [PubMed: 26000488]
3. Rosenberg AB et al. Single-cell profiling of the developing mouse brain and spinal cord with split-pool barcoding. Science 360, 176–182 (2018). [PubMed: 29545511]
4. Cao J et al. The single-cell transcriptional landscape of mammalian organogenesis. Nature 566, 496–502 (2019). [PubMed: 30787437]

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

5. Yang A, Troup M, Lin P & Ho JWK Falco: a quick and flexible single-cell RNA-seq processing framework on the cloud. Bioinformatics 33, 767–769 (2017). [PubMed: 28025200]
6. Census of Immune Cells (data from Immune Cell Atlas). https://data.humancellatlas.org/explore/projects/cc95ff89-2e68-4a08-a234-480eca21ce79.
7. Zheng GXY et al. Massively parallel digital transcriptional profiling of single cells. Nat. Commun. 8, 14049 (2017). [PubMed: 28091601]
8. Picelli S et al. Smart-seq2 for sensitive full-length transcriptome profiling in single cells. Nat. Methods 10, 1096–1098 (2013). [PubMed: 24056875]
9. Stoeckius M et al. Simultaneous epitope and transcriptome measurement in single cells. Nat. Methods 14, 865–868 (2017). [PubMed: 28759029]
10. Stoeckius M et al. Cell hashing with barcoded antibodies enables multiplexing and doublet detection for single cell genomics. Genome Biol. 19, 224 (2018). [PubMed: 30567574]
11. Gaublomme JT et al. Nuclei multiplexing with barcoded antibodies for single-nucleus genomics. Nat. Commun. 10, 2907 (2019). [PubMed: 31266958]
12. Dixit A et al. Perturb-seq: dissecting molecular circuits with scalable single-cell rna profiling of pooled genetic screens. Cell 167, 1853–1866.e17 (2016). [PubMed: 27984732]
13. Datlinger P et al. Pooled CRISPR screening with single-cell transcriptome readout. Nat. Methods 14, 297–301 (2017). [PubMed: 28099430]
14. Coifman RR & Lafon S Diffusion maps. Applied and Computational Harmonic Analysis 21, 5–30 (2006).
15. Haghverdi L, Büttner M, Wolf FA, Buettner F & Theis FJ Diffusion pseudotime robustly reconstructs lineage branching. Nat. Methods 13, 845 (2016). [PubMed: 27571553]
16. Blondel VD, Guillaume J-L, Lambiotte R & Lefebvre E Fast unfolding of communities in large networks. J. Stat. Mech.: Theory Exp 10, P10008 (2008).
17. Traag VA, Waltman L & Eck N. J. van. From Louvain to Leiden: guaranteeing well-connected communities. Sci. Rep 9, 5233 (2019). [PubMed: 30914743]
18. Maaten L. van der & Hinton G Visualizing data using t-SNE. J. Mach. Learn. Res. 9, 2579–2605 (2008).
19. Linderman GC, Rachh M, Hoskins JG, Steinerberger S & Kluger Y Fast interpolation-based t-SNE for improved visualization of single-cell RNA-seq data. Nat. Methods 16, 243 (2019). [PubMed: 30742040]
20. McInnes L, Healy J & Melville J UMAP: uniform manifold approximation and projection for dimension reduction. arXiv:1802.03426v2 [stat.ML] (2018).
21. Becht E et al. Dimensionality reduction for visualizing single-cell data using UMAP. Nat. Biotechnol. 37, 38–44 (2019).
22. Jacomy M, Venturini T, Heymann S & Bastian M ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software. PLOS ONE 9, e98679 (2014). [PubMed: 24914678]
23. Schiebinger G et al. Optimal-transport analysis of single-cell gene expression identifies developmental trajectories in reprogramming. Cell 176, 928–943.e22 (2019). [PubMed: 30712874]
24. Tabaka M, Gould J & Regev A scSVA: an interactive tool for big data visualization and exploration in single-cell omics. bioRxiv 512582 (2019) doi:10.1101/512582.
25. Ke G et al. LightGBM: a highly efficient gradient boosting decision tree. in Advances in Neural Information Processing Systems 30 3146–3154 (2017).
26. Satija R, Farrell JA, Gennert D, Schier AF & Regev A Spatial reconstruction of single-cell gene expression data. Nat. Biotechnol. 33, 495–502 (2015). [PubMed: 25867923]
27. Wolf FA, Angerer P & Theis FJ SCANPY: large-scale single-cell gene expression data analysis. Genome Biol. 19, 15 (2018). [PubMed: 29409532]
28. Bhattacharya S et al. ImmPort, toward repurposing of open access immunological assay data for translational and clinical research. Sci. Data 5, 180015 (2018). [PubMed: 29485622]
29. Li C & Wong WH DNA-chip analyzer (dChip). in The Analysis of Gene Expression Data 120–141 (2003).

30. Johnson WE, Li C & Rabinovic A Adjusting batch effects in microarray expression data using empirical Bayes methods. Biostatistics 8, 118–127 (2007). [PubMed: 16632515]
31. Haghverdi L, Lun ATL, Morgan MD & Marioni JC Batch effects in single-cell RNA-sequencing data are corrected by matching mutual nearest neighbors. Nat. Biotechnol. 36, 421 (2018). [PubMed: 29608177]
32. Pola ski K et al. BBKNN: fast batch alignment of single cell transcriptomes. Bioinformatics 36, 964–965 (2020). [PubMed: 31400197]
33. Stuart T et al. Comprehensive integration of single-cell data. Cell 177, 1888–1902.e21 (2019). [PubMed: 31178118]
34. Büttner M, Miao Z, Wolf FA, Teichmann SA & Theis FJ A test metric for assessing single-cell RNA-seq batch correction. Nat. Methods 16, 43 (2019). [PubMed: 30573817]
35. Malkov YA & Yashunin DA Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. IEEE Trans. Pattern Anal. Mach. Intell. 42, 824–836 (2020). [PubMed: 30602420]
36. Aumüller M, Bernhardsson E & Faithfull A ANN-benchmarks: a benchmarking tool for approximate nearest neighbor algorithms. in Similarity Search and Applications 34–49 (2017).
37. Alavi A, Ruffalo M, Parvangada A, Huang Z & Bar-Joseph Z A web server for comparative analysis of single-cell RNA-seq data. Nat. Commun 9, 1–11 (2018). [PubMed: 29317637]
38. van Dijk D et al. Recovering gene interactions from single-cell data using data diffusion. Cell 174, 716–729.e27 (2018). [PubMed: 29961576]
39. Moon KR et al. Visualizing structure and transitions in high-dimensional biological data. Nat. Biotechnol 37, 1482–1492 (2019). [PubMed: 31796933]
40. Anand K, Bianconi G & Severini S Shannon and von Neumann entropy of random networks with heterogeneous expected degree. Phys. Rev. E 83, 036109 (2011).
41. Luxburg U von. A tutorial on spectral clustering. Stat. Comput. 17, 395–416 (2007).
42. Cho H, Berger B & Peng J Generalizable and scalable visualization of single-cell data using neural networks. Cell Syst. 7, 185–191.e4 (2018). [PubMed: 29936184]
43. Srivastava A, Malik L, Smith T, Sudbery I & Patro R Alevin efficiently estimates accurate gene abundances from dscRNA-seq data. Genome Biol. 20, 1–16 (2019). [PubMed: 30606230]
44. Melsted P et al. Modular and efficient pre-processing of single-cell RNA-seq. bioRxiv 673285 (2019) doi:10.1101/673285.
45. Slyper M et al. A single-cell and single-nucleus RNA-Seq toolbox for fresh and frozen human tumors. Nat. Med. 26(5), 792–802 (2020). [PubMed: 32405060]

References

46. Guo R, Zhao Y, Zou Q, Fang X & Peng S Bioinformatics applications on Apache Spark. Gigascience 7, (2018).
47. Petukhov V et al. dropEst: pipeline for accurate estimation of molecular counts in droplet-based single-cell RNA-seq experiments. Genome Biol. 19, 1–16 (2018). [PubMed: 29301551]
48. Li B & Dewey CN RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. BMC Bioinform. 12, 323 (2011).
49. Kim D, Paggi JM, Park C, Bennett C & Salzberg SL Graph-based genome alignment and genotyping with HISAT2 and HISAT-genotype. Nat Biotechnol 37, 907–915 (2019). [PubMed: 31375807]
50. Langmead B & Salzberg SL Fast gapped-read alignment with Bowtie 2. Nat. Methods 9, 357–359 (2012). [PubMed: 22388286]
51. Dobin A et al. STAR: ultrafast universal RNA-seq aligner. Bioinformatics 29, 15–21 (2013). [PubMed: 23104886]
52. Dixit A Correcting chimeric crosstalk in single cell RNA-seq experiments. bioRxiv 093237 (2016) doi:10.1101/093237.
53. Cleveland WS, Grosse E & Shyu WM Local regression models. in Statistical Models in S (1992).

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Author Manuscript

Author Manuscript

Author Manuscript

Author Manuscript

54. Halko N, Martinsson PG & Tropp JA Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. SIAM Rev. 53, 217–288 (2011).

55. Pedregosa F et al. Scikit-learn: machine learning in Python. J. Mach. Learn. Res. 12, 2825–2830 (2011).

56. Calvetti D, Reichel L & Sorensen DC An implicitly restarted lanczos method for large symmetric eigenvalue problems. Electron. Trans. Numer. Anal. 2, 1–21 (1994).

57. Reichardt J & Bornholdt S Statistical mechanics of community detection. Phys. Rev. E 74, 016110 (2006).

58. Benjamini Y & Hochberg Y Controlling the false discovery rate: a practical and powerful approach to multiple testing. J. R. Stat. Soc. B Met. 57, 289–300 (1995).

59. Vinh NX, Epps J & Bailey J Information theoretic measures for clusterings comparison: is a correction for chance necessary? in Proceedings of the 26th Annual International Conference on Machine Learning 1073–1080 (2009).

60. Natarajan KN et al. Comparative analysis of sequencing technologies for single-cell transcriptomics. Genome Biol 20, 1–8 (2019). [PubMed: 30606230]

Author Manuscript
Author Manuscript
Author Manuscript
Author Manuscript

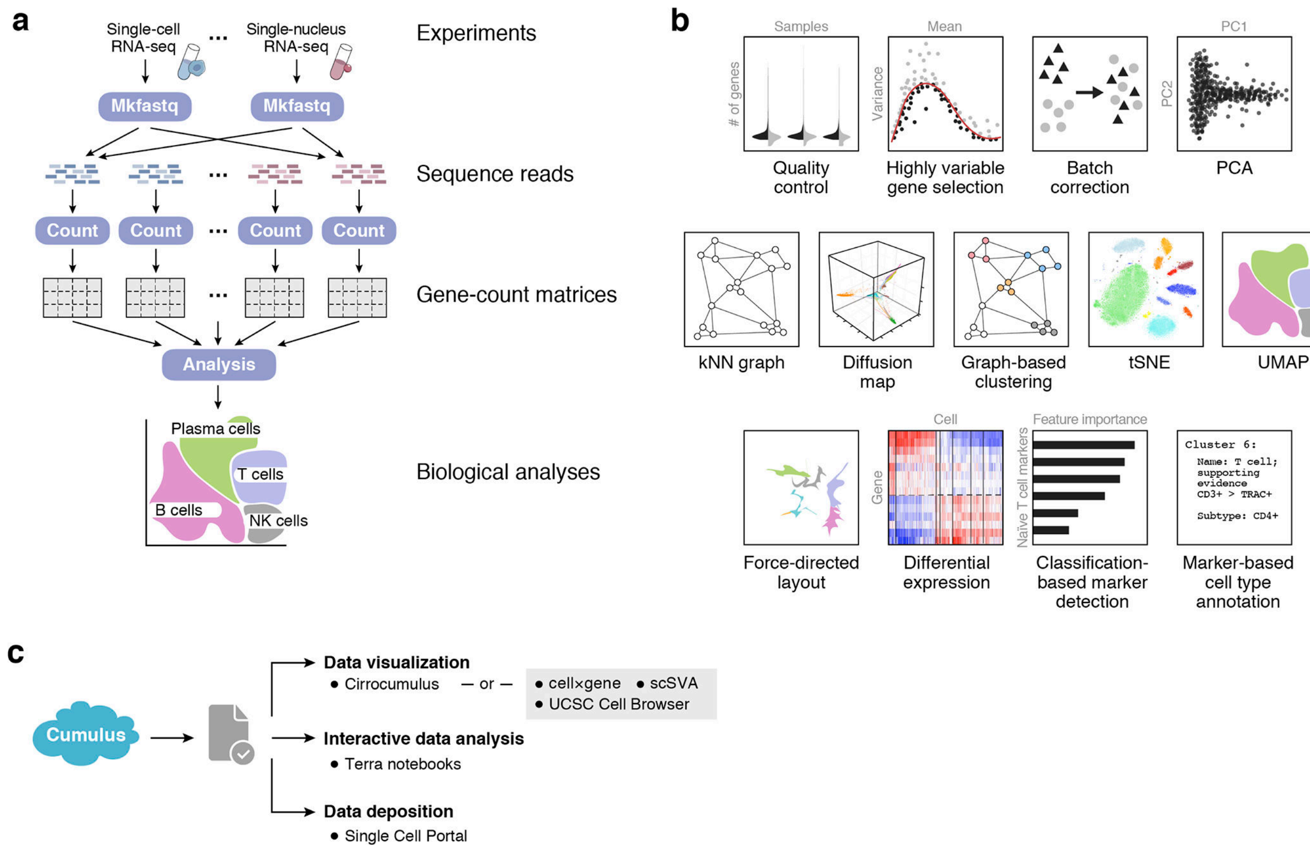


Figure 1. Cumulus: a scalable, feature-rich, accessible cloud-based framework for sc/sn RNA-seq analysis.

a. Cumulus data analysis workflow. Cumulus takes raw BCL files as input and outputs diverse analysis results, with three key computational steps – *mkfastq*, *count*, and *analysis*. **b.** sc/snRNA-seq analysis tasks in Pegasus. **c.** Cumulus enables flexible interactive data visualization and analysis. Users can instantly visualize Cumulus analysis results with Cirrocumulus, or publicly available visualization tools such as cellxgene, UCSC cell browser and scSVA. They can also interactively explore them on Terra Jupyter notebooks using Pegasus and deposit their data into the Single Cell Portal.

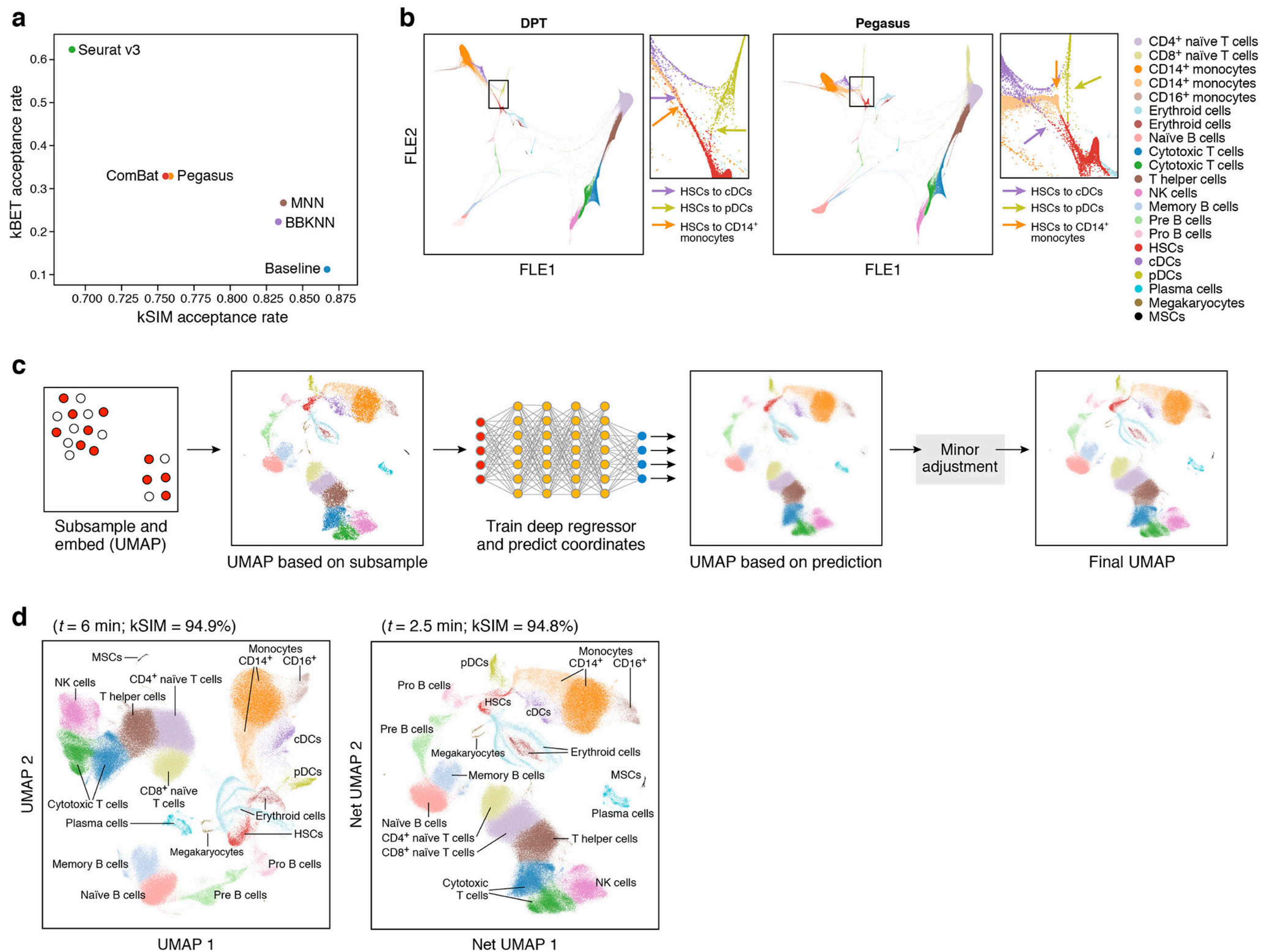


Figure 2. Algorithmic and implementation improvements underlying Pegasus's high scalability.
a. Trade-off between kBET and kSIM acceptance rates across different methods. kBET (*y* axis) and kSIM (*x* axis) acceptance rates of Pegasus, ComBat, MNN, BBKNN and Seurat v3 on 34,654 bone marrow cells. **b.** Improved resolution of a developmental bifurcation with diffusion pseudotime map with timescale selected by von Neumann entropy (n = 274,182 bone marrow cells). Diffusion maps of cells colored by subset annotation (color legend), generated by DPT (left) and Pegasus (right). Red square: area of bifurcation from hematopoietic stem cells (HSCs) to *CD14*+ monocytes (orange arrow) and conventional dendritic cells (cDCs, purple arrow) (zoom, right), in each map. **c.** Deep-learning-based efficient visualization with Net-*. From left: a small fraction of cells is subsampled based on local density and then embedded (*e.g.*, with UMAP); a deep regressor is trained on the subsampled cells to predict the embedding coordinates; it is then used to predict embedding coordinates for remaining cells; all the coordinates are fine-tuned by applying the embedding algorithm (*e.g.*, UMAP) for a small number of iterations. **d.** Net-UMAP visualization is faster than UMAP while maintaining visualization quality (n = 274,182 bone marrow cells). Embedding generated by UMAP (left) and Net-UMAP (right) of cells, colored by subset annotation. Top: Execution time and kSIM acceptance rate.

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Author Manuscript
Author Manuscript
Author Manuscript
Author Manuscript

Table 1.

Cumulus is computationally efficient and cost effective

	Time			Cost per sample
Step	Cell Ranger + Seurat v3	Cell Ranger + SCANPY	Cumulus	Cumulus
Total	**10 d, 5 h, 38 min**	**9 d, 5 h, 35 min**	**15 h, 15 min**	**$1.832**
Mkfastq	13 h, 18 min	13 h, 18 min	7 h, 54 min	$0.22
Count	8 d, 14 h, 12 min	8 d, 14 h, 12 min	6 h, 44 min	$1.61
Analysis	26 h, 8 min	2 h, 5 min	37 min	$0.002

Left columns: Total execution time on the bone marrow dataset of Cumulus, Cell Ranger + Seurat v3 or Cell Ranger + SCANPY pipeline, running on a 32 CPU thread, 120 GB memory Google Cloud virtual machine instance (Methods). Right columns: Average computational cost for running Cumulus per sample of ~4,000 cells (Methods).