nature methods

ARTICLES

https://doi.org/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]✉ and Aviv Regev[1,7,8,9]✉

Massively parallel single-cell and single-nucleus RNA sequencing has opened the way to systematic tissue atlases in health and disease, but as the scale of data generation is growing, so is the need for computational pipelines for scaled analysis. Here we developed Cumulus—a cloud-based framework for analyzing large-scale single-cell and single-nucleus RNA sequencing datasets. Cumulus combines the power of cloud computing with improvements in algorithm and implementation 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.

Single-cell and single-nucleus RNA sequencing (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]. However, such large and growing datasets pose a critical 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 support 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 and single-nucleus genomics. Cumulus consists of a cloud analysis workflow, a Python analysis package (Pegasus) and a visualization application (Cirrocumulus) (Fig. 1). Cumulus performs three major steps in sc/snRNA-seq data analysis (Fig. 1a): (**1**) sequence read extraction; (**2**) gene-count matrix generation; and (**3**) biological analysis. 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 (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 (HVG) selection, batch correction, inference of pseudotemporal trajectories, clustering and data visualization. To test Cumulus and compare it with other tools, we relied on an scRNA-seq dataset of 274,182 cells (Methods), which were profiled from the bone marrow of eight 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 and single-nucleus genomics. Cloud computing offers on-demand scalable computing, high-availability storage, data security and installation-free software-as-a-service 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 (that is, sequence read extraction and gene-count matrix generation) in parallel across a large number of computer nodes, and executes the last step of analysis in a single multi-central processing unit (CPU) node, using Pegasus (Methods). Non-programming biologist users readily access computing resources on the cloud through a simple web-based user interface provided by Terra (Supplementary Video 1).

Cumulus and Pegasus support analysis starting from a variety of input modalities. Given the growing diversity of laboratory protocols and experimental designs 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

[1]Klarman Cell Observatory, Broad Institute of Harvard and MIT, Cambridge, MA, USA. [2]Division of Rheumatology, Allergy, and Immunology, Center for Immunology and Inflammatory Diseases, 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. ✉e-mail: bli28@mgh.harvard.edu; orit@broadinstitute.org; aregev@broadinstitute.org

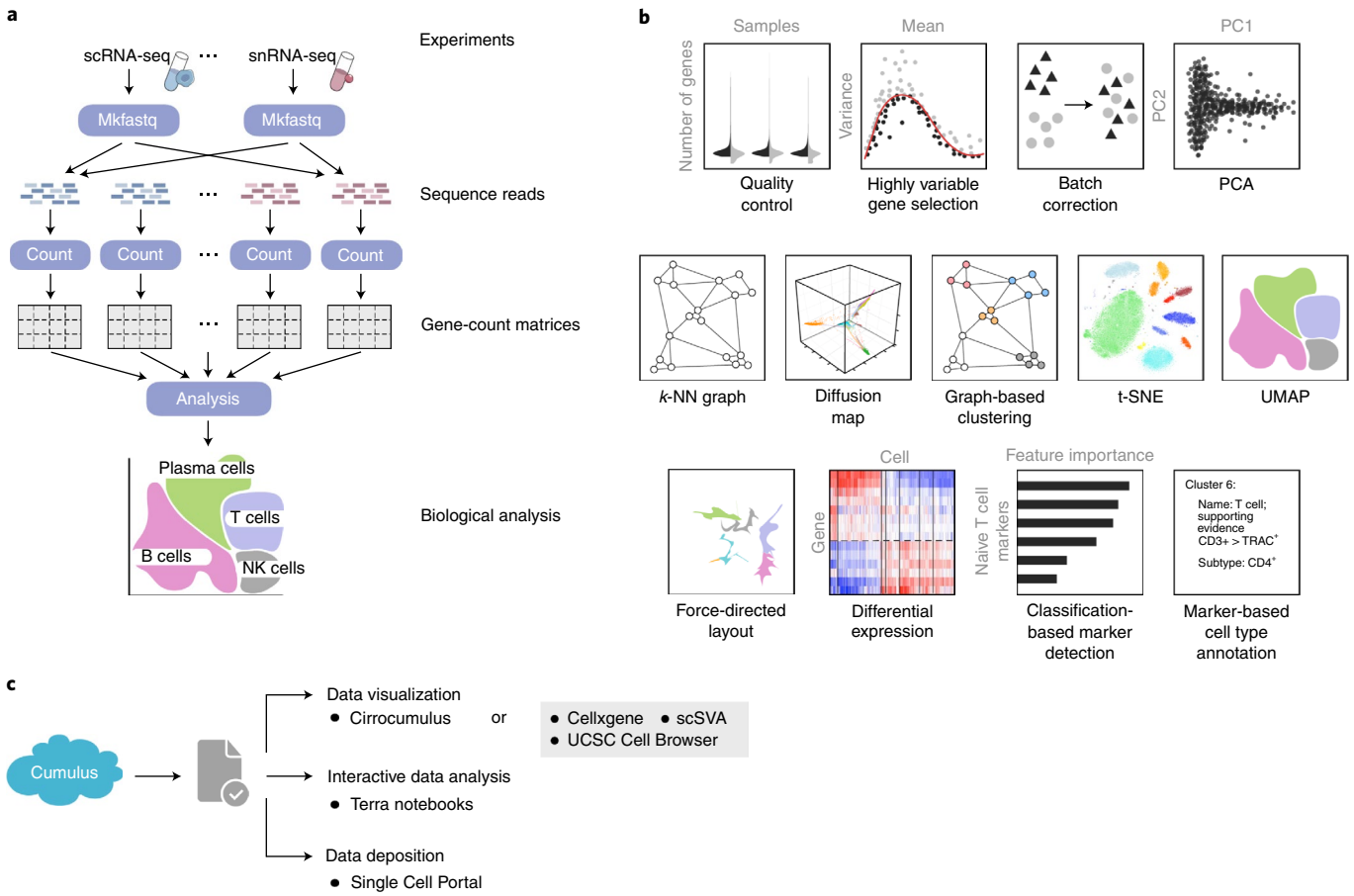


Fig. 1 | Cumulus: a scalable, feature-rich, accessible cloud-based framework for sc/snRNA-seq analysis. a, Cumulus data analysis workflow. Cumulus takes raw base call files as input and outputs diverse analysis results, with three key computational steps: mkfastq, count and analysis. **b**, sc/snRNA-seq 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. NK, natural killer.

transcriptome as a core readout (Supplementary Table 2). These include: droplet-based[2,7] (3′ or 5′ ends; with unique molecular identifiers (UMIs)) and plate-based[8] sc/snRNA-seq (full length; no UMIs) (Methods); cellular indexing of transcriptomes and epitopes by sequencing (CITE-seq)[9], which simultaneously measures messenger RNA expression and the abundance of oligo-tagged surface antibodies in single cells (Methods); data from cell[10] or nucleus[11] hashing experiments, which are laboratory 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 (Fig. 1b and Methods). Starting from a gene-count matrix, Pegasus filters out low-quality cells/nuclei, selects HVGs 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 space, calculates diffusion maps[14,15] and applies community detection algorithms on the graph to find clusters[16,17]. It visualizes cell profiles using methods based on either t-distributed stochastic neighbor embedding (t-SNE)[18,19] or uniform manifold approximation and projection (UMAP)[20,21]. It can additionally estimate diffusion pseudotime (DPT)[15] and visualize developmental trajectories using algorithms based on force-directed layout embedding (FLE)[22–24]. Pegasus can be used to detect cluster-specific markers, by differential expression analysis between cells within and outside a cluster and optionally calculates the area under the receiver operating 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 (single-cell analysis in Python)[27], although each package also has some unique features that are 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 (Fig. 1c and 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 scSVA[24] (Fig. 1c). Alternatively, users can inspect or re-analyze their data interactively using Pegasus on Terra Jupyter Notebooks (Fig. 1c and 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 (Fig. 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 HVGs, batch correction, *k*-NN graph construction, calculation of DPT, a combination of spectral and community-based clustering, and efficient visualization algorithms. We describe each of these enhancements in turn, comparing its impacts on analysis quality and analysis speed/scale with those of 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 Fig. 1a and 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 versus the standard one. It recovers slightly more immune-specific genes provided by the ImmPort[28] data repository (Extended Data Fig. 1b, Supplementary Data 1 and 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 are missed by the standard procedure (Extended Data Fig. 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' L/S method, SCANPY's offerings of ComBat[30], MNN[31] and BBKNN[32], and the most recent integration method[33] of Seurat version 3 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 whether batches are well-mixed in the local neighborhood of each cell, whereas the kSIM acceptance rate measures whether 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 whether known biological relationships are preserved after correction. An ideal batch-correction method should have both high kBET and high kSIM acceptance rates. Each of the five methods evaluated showed a trade-off between the two rates, and none was a clear best performer (Fig. 2a). Pegasus' L/S method is the fastest (Extended Data Fig. 2a), while maintaining a good balance between the two rates (Fig. 2a and Extended Data Fig. 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-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 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 Fig. 3a), while being 3–19× faster (Extended Data Fig. 3b). HNSW was also benchmarked recently for plate-based, small-scale scRNA-seq[37].

To speed up the calculation of diffusion maps[14] and DPT[15], we adopted two modifications that are also used by SCANPY and scSVA and further improved the identification of pseudotemporal trajectories[15] with two additional modifications. As in SCANPY, we constructed the affinity matrix based on the approximate *k*-NN graph constructed in the previous step instead of a complete graph, and also used only the top *n* diffusion components to approximate diffusion distances and thus DPTs, where *n* is a user-specific parameter. In addition, to better identify pseudotemporal trajectories when there were multiple subsets of cells undergoing separate temporal processes, we found that using more diffusion components helped us to better separate different cell populations (red regions in Extended Data Fig. 4a); thus, we set $n = 100$ by default. We further introduced a family of 'diffusion pseudotime maps' parameterized by timescale *t*: each pseudotime meta-map was 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 chose 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 Fig. 4b and Methods). In the bone marrow data, using the selected *t*, we identified a trajectory (Fig. 2b) that more clearly bifurcates from hematopoietic stem cells into $CD14^+$ monocytes and conventional dendritic cells, 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 DPT components is very fast, but its clustering results are not always satisfactory (Extended Data Fig. 5a). Conversely, in Pegasus' 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 Fig. 5b,c).

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 FLE[23] of the DPT map (Methods), we also included 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 a higher rate of sampling from rare and sparse clusters, and then embeds the subsampled cells using the embedding algorithm of interest, such as UMAP (Fig. 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

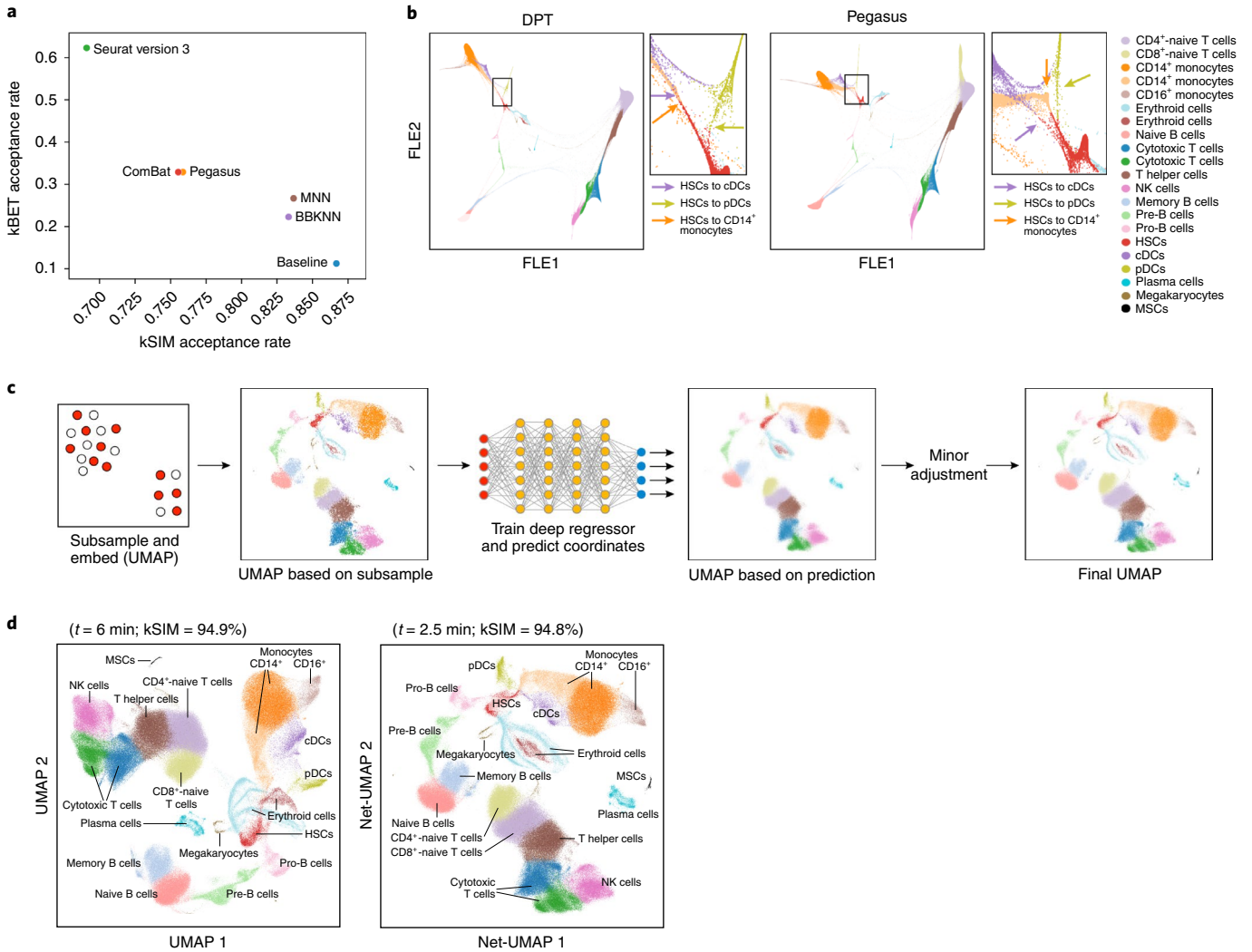


Fig. 2 | Algorithmic and implementation improvements underlying Pegasus' high scalability. a, Trade-off between kBET and kSIM acceptance rates across different methods. The kBET (*y* axis) and kSIM (*x* axis) acceptance rates of Pegasus, ComBat, MNN, BBKNN and Seurat version 3 on 34,654 bone marrow cells are shown. **b**, Improved resolution of a developmental bifurcation with DPT 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), are shown. For each map, the area marked by a rectangle shows the area of bifurcation from hematopoietic stem cells (HSCs) to CD14$^+$ monocytes (orange arrow), conventional dendritic cells (cDCs, purple arrow) and plasmacytoid dendritic cells (pDCs; yellow arrow) (magnified in the plots to the right). **c**, Deep-learning-based efficient visualization with Net-*. From left: a small fraction of cells are subsampled based on local density and then embedded (for example, 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 the remaining cells; and all of the coordinates are fine-tuned by applying the embedding algorithm (for example, 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 is shown, colored by subset annotation. Execution times and kSIM acceptance rates are shown above each plot. MSCs: mesenchymal stem cells.

remaining cells. Because the predicted coordinates yield a blurry visualization (Fig. 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 (Fig. 2c). We call visualizations obtained using this technique Net-* visualizations and showed that they speed up the original embedding algorithm by at least 2× and maintain the visualization quality based on similar kSIM acceptance rates (Fig. 2d and Extended Data Fig. 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 analysis tasks on the bone marrow dataset[6] (Supplementary Table 4 and 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 and Methods). We validated that these improvements run efficiently on both small and large datasets with different thread settings by subsampling the bone marrow dataset (Supplementary Data 2 and Methods), and demonstrated that Pegasus worked equally well for small datasets on a 10x Genomics 5K peripheral blood mononuclear cell (PBMC) dataset (https://support.10xgenomics.com/single-cell-gene-expression/datasets/3.0.2/5k_pbmc_v3) (Supplementary Table 6 and Methods). Furthermore, Pegasus is memory efficient compared with other packages, based on peak memory use on the 5K PBMC and bone marrow datasets (Supplementary Table 7 and Methods).

Table 1 | Cumulus is computationally efficient and cost-effective

	Time[a]			Cost per sample[b]
	Cell Ranger + Seurat version 3	Cell Ranger + SCANPY	Cumulus	Cumulus
Total	10 d, 5 h, 38 min	9 d, 5 h, 35 min	15 h, 15 min	US$1.832
Mkfastq	13 h, 18 min	13 h, 18 min	7 h, 54 min	US$0.22
Count	8 d, 14 h, 12 min	8 d, 14 h, 12 min	6 h, 44 min	US$1.61
Analysis	26 h, 8 min	2 h, 5 min	37 min	US$0.002

[a]Total execution time on the bone marrow dataset of the Cumulus, Cell Ranger + Seurat version 3 or Cell Ranger + SCANPY pipeline, running on a 32-CPU-thread, 120-GB-memory Google Cloud virtual machine instance (Methods). [b]Average computational cost for running Cumulus per sample of ~4,000 cells (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 with a Cell Ranger + Seurat/SCANPY pipeline, Cumulus completed the analysis in around 15 h, while the alternative pipeline took over 9 d to run (Table 1 and Methods). The associated computational costs were modest (for example, ~US$2 on average for around 4,000 cells in one sample; Table 1 and 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 Fig. 7, Supplementary Data 3 and 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 scRNA-seq data (Supplementary Data 4 and Methods). Besides Cell Ranger, Cumulus also supports gene-count matrix generation using Optimus (the Human Cell Atlas 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 the cloud-based performance of Cell Ranger, Optimus, Alevin, Kallisto-BUStools and STARsolo on the 5K PBMC dataset (Supplementary Table 8 and Methods).

Discussion

Cumulus provides the community with a cloud-based, scalable, cost-effective, comprehensive and easy-to-use platform for sc/snRNA-seq research. Pegasus—Cumulus' analysis module, which can also be used as an independent Python package—implements many improvements that 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 laboratories to analyze large-scale single-cell and single-nucleus datasets. In a separate study[45], we demonstrated Cumulus by analyzing sc/snRNA-seq data from fresh and frozen tumors from the Human Tumor Atlas Pilot Project. As the community produces datasets 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 in leveraging them to understand the human body in health and disease.

Online content

Any methods, additional references, Nature Research reporting summaries, source data, extended data, supplementary information, acknowledgements, peer review information; details of author contributions and competing interests; and statements of data and code availability are available at https://doi.org/10.1038/s41592-020-0905-x.

Received: 29 October 2019; Accepted: 18 June 2020;
Published online: 27 July 2020

References

1. Regev, A. et al. The Human Cell Atlas White Paper. Preprint at https://arxiv.org/abs/1810.05192 (2018).
2. Macosko, E. Z. et al. Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets. *Cell* **161**, 1202–1214 (2015).
3. Rosenberg, A. B. et al. Single-cell profiling of the developing mouse brain and spinal cord with split-pool barcoding. *Science* **360**, 176–182 (2018).
4. Cao, J. et al. The single-cell transcriptional landscape of mammalian organogenesis. *Nature* **566**, 496–502 (2019).
5. Yang, A., Troup, M., Lin, P. & Ho, J. W. K. Falco: a quick and flexible single-cell RNA-seq processing framework on the cloud. *Bioinformatics* **33**, 767–769 (2017).
6. Kowalczyk, M. S. et al. *Census of Immune Cells* (Human Cell Atlas). https://data.humancellatlas.org/explore/projects/cc95ff89-2e68-4a08-a234-480eca21ce79 (2018).
7. Zheng, G. X. Y. et al. Massively parallel digital transcriptional profiling of single cells. *Nat. Commun.* **8**, 14049 (2017).
8. Picelli, S. et al. Smart-seq2 for sensitive full-length transcriptome profiling in single cells. *Nat. Methods* **10**, 1096–1098 (2013).
9. Stoeckius, M. et al. Simultaneous epitope and transcriptome measurement in single cells. *Nat. Methods* **14**, 865–868 (2017).
10. Stoeckius, M. et al. Cell hashing with barcoded antibodies enables multiplexing and doublet detection for single cell genomics. *Genome Biol.* **19**, 224 (2018).
11. Gaublomme, J. T. et al. Nuclei multiplexing with barcoded antibodies for single-nucleus genomics. *Nat. Commun.* **10**, 2907 (2019).
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).
13. Datlinger, P. et al. Pooled CRISPR screening with single-cell transcriptome readout. *Nat. Methods* **14**, 297–301 (2017).
14. Coifman, R. R. & Lafon, S. Diffusion maps. *Appl. Comput. Harmon. Anal.* **21**, 5–30 (2006).
15. Haghverdi, L., Büttner, M., Wolf, F. A., Buettner, F. & Theis, F. J. Diffusion pseudotime robustly reconstructs lineage branching. *Nat. Methods* **13**, 845–848 (2016).
16. Blondel, V. D., Guillaume, J.-L., Lambiotte, R. & Lefebvre, E. Fast unfolding of communities in large networks. *J. Stat. Mech. Theory Exp.* **10**, P10008 (2008).
17. Traag, V. A., Waltman, L. & van Eck, N. J. From Louvain to Leiden: guaranteeing well-connected communities. *Sci. Rep.* **9**, 5233 (2019).
18. Van der Maaten, L. & Hinton, G. Visualizing data using t-SNE. *J. Mach. Learn. Res.* **9**, 2579–2605 (2008).
19. Linderman, G. C., Rachh, M., Hoskins, J. G., Steinerberger, S. & Kluger, Y. Fast interpolation-based t-SNE for improved visualization of single-cell RNA-seq data. *Nat. Methods* **16**, 243–245 (2019).
20. McInnes, L., Healy, J. & Melville, J. UMAP: uniform manifold approximation and projection for dimension reduction. Preprint at https://arxiv.org/abs/1802.03426 (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).
23. Schiebinger, G. et al. Optimal-transport analysis of single-cell gene expression identifies developmental trajectories in reprogramming. *Cell* **176**, 928–943.e22 (2019).
24. Tabaka, M., Gould, J. & Regev, A. scSVA: an interactive tool for big data visualization and exploration in single-cell omics. Preprint at *bioRxiv* https://doi.org/10.1101/512582 (2019).
25. Ke, G. et al. LightGBM: a highly efficient gradient boosting decision tree. *Adv. Neural Inform. Process. Syst.* **30**, 3146–3154 (2017).
26. Satija, R., Farrell, J. A., Gennert, D., Schier, A. F. & Regev, A. Spatial reconstruction of single-cell gene expression data. *Nat. Biotechnol.* **33**, 495–502 (2015).

27. Wolf, F. A., Angerer, P. & Theis, F. J. SCANPY: large-scale single-cell gene expression data analysis. *Genome Biol.* **19**, 15 (2018).
28. Bhattacharya, S. et al. ImmPort, toward repurposing of open access immunological assay data for translational and clinical research. *Sci. Data* **5**, 180015 (2018).
29. Li, C. & Wong, W. H. DNA-Chip analyzer (dChip). in *The Analysis of Gene Expression Data* 120–141 (Springer, 2003).
30. Johnson, W. E., Li, C. & Rabinovic, A. Adjusting batch effects in microarray expression data using empirical Bayes methods. *Biostatistics* **8**, 118–127 (2007).
31. Haghverdi, L., Lun, A. T. L., Morgan, M. D. & Marioni, J. C. Batch effects in single-cell RNA-sequencing data are corrected by matching mutual nearest neighbors. *Nat. Biotechnol.* **36**, 421–427 (2018).
32. Polański, K. et al. BBKNN: fast batch alignment of single cell transcriptomes. *Bioinformatics* **36**, 964–965 (2020).
33. Stuart, T. et al. Comprehensive integration of single-cell data. *Cell* **177**, 1888–1902.e21 (2019).
34. Büttner, M., Miao, Z., Wolf, F. A., Teichmann, S. A. & Theis, F. J. A test metric for assessing single-cell RNA-seq batch correction. *Nat. Methods* **16**, 43–49 (2019).
35. Malkov, Y. A. & Yashunin, D. A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* **42**, 824–836 (2020).
36. Aumüller, M., Bernhardsson, E. & Faithfull, A. ANN-benchmarks: a benchmarking tool for approximate nearest neighbor algorithms. in *Similarity Search and Applications* 34–49 (Springer, 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**, 4768 (2018).
38. Van Dijk, D. et al. Recovering gene interactions from single-cell data using data diffusion. *Cell* **174**, 716–729.e27 (2018).
39. Moon, K. R. et al. Visualizing structure and transitions in high-dimensional biological data. *Nat. Biotechnol.* **37**, 1482–1492 (2019).
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. Von Luxburg, U. 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).
43. Srivastava, A., Malik, L., Smith, T., Sudbery, I. & Patro, R. Alevin efficiently estimates accurate gene abundances from dscRNA-seq data. *Genome Biol.* **20**, 65 (2019).
44. Melsted, P. et al. Modular and efficient pre-processing of single-cell RNA-seq. Preprint at *bioRxiv* https://doi.org/10.1101/673285 (2019).
45. Slyper, M. et al. A single-cell and single-nucleus RNA-Seq toolbox for fresh and frozen human tumors. *Nat. Med.* **26**, 792–802 (2020).

Methods

Cumulus modules. *Gene-count matrix generation for droplet-based scRNA-seq.* Cumulus supports gene-count matrix generation for 10x Genomics version 2 and version 3 chemistry using Cell Ranger. Cumulus first demultiplexes Illumina base call files for each sequencing flow cell by running mkfastq steps in parallel 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 10× channel by running count steps in parallel. Each count job calls cellranger count with the 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-seq[2] data using either the methods described in the 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 (ref. [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 (ref. [50]), or to a reference genome using STAR[51]. Each node requests four CPUs and 3.6 GB memory by default if utilizing HISAT2 or Bowtie 2 aligner, and requests four 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 transcripts per 100,000 (TP100K)) into a count vector for each single cell using the following formula and then generates a gene-count matrix by concatenating count vectors from all cells.

$$c_i = \frac{\mathrm{TP100K}_i}{10^5} \cdot c_{\mathrm{tot}}, \tag{1}$$

where c_i and $\mathrm{TP100K}_i$ 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 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 hashing[11] and Perturb-seq[12,13] protocols, using either 10x Genomics version 2 or version 3 chemistry. Each feature-count matrix generation job runs in parallel on a separate compute node with one 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 that read 1 records the cellular barcode and UMI information and read 2 records feature barcode information, such as hash tags for hashing protocols or single guide RNA information for Perturb-seq (below). The first 16 nucleotides of read 1 represent the cell barcode for both version 2 and version 3 chemistry. The next ten and 12 nucleotides represent the UMI for version 2 and version 3 chemistry, respectively. We allow up to one and zero mismatches for matching cell barcodes in version 2 and version 3 chemistry, respectively.

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 the type of BioLegend TotalSeq antibodies that 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 eleventh nucleotide from the 5′ end of read 2. The program generate_count_matrix_ADTs automatically detects the antibody type by scanning read 2 of the first 10,000 read pairs and calculating the percentage of read pairs containing feature barcodes at the 5′ end of read 2. If more than 50% of read pairs contain feature barcodes at the 5′ end of read 2, we assume the antibody type is TotalSeq-A; otherwise, it is TotalSeq-B or TotalSeq-C.

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 two 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 three mismatches by default. After scanning all of the 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 (two 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 immunoglobulin G (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:

$$\mathrm{expr} = \max\left(\log\left[\frac{c_a + 1}{c_c + 1}\right], 0\right), \tag{2}$$

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

$$\mathrm{expr} = \log[c_a + 1] \tag{3}$$

Cumulus merges the transformed antibody expression matrix into an RNA expression matrix so that users can plot antibody expression in two-dimensional (2D) visualizations (for example, t-SNE and 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, single guide RNAs 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 ($\exists i, c[i-2] < c[i-1] < c[i] \geq c[i+1]$), 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 $i \leq 10$, 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 one 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 whether 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 extracts reads from each Illumina flow cell in different nodes in parallel. In most cases, users sequence their libraries using one type of sequencer (for example, 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 a 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; 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 2 million 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 the Google App Engine application for collaborative use or it 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 the 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 and 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 the 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. The plots provided include scatter plots, feature plots, dot plots and violin plots and can scale to millions of cells by plotting cells on a 2D grid. scPlot uses HoloViews (http://holoviews.org/), thus allowing generation of interactive plots with Bokeh for a Jupyter notebook.

Preprocessing. Pegasus selects high-quality cells based on a combination of the following criteria, with user-provided parameters: (1) number of UMIs between (min_umis, max_umis), with default min_umis = 100 and max_umis = 600,000; (2) number of expressed genes (at least one UMI) between (min_genes, max_genes), with default min_genes = 500 and max_genes = 6,000; and (3) percentage of UMIs from mitochondrial genes less than percent_mito, with 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), with default $x = 0.05\%$ (equivalent to three 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 (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.

HVG 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 developed a new HVG selection procedure that operates directly on 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:

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

We then fit a LOESS[53] curve of degree 2 (span parameter: 0.02) between the estimated means and variances (Extended Data Fig. 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.

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

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

We then rank each robust gene with respect to δ_g and τ_g in descending order, and denote their rankings as $\text{rank}_\delta(g)$ and $\text{rank}_\tau(g)$, respectively. Lastly, we define the overall ranking as the sum of the two rankings:

$$\text{rank}(g) = \text{rank}_\delta(g) + \text{rank}_\tau(g) \tag{6}$$

and select the top n robust genes with respect to $\text{rank}(g)$ 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 (batch j in group k) has n_{kj} cells. We additionally denote the mean of gene g within batch kj and within group k as $\hat{\mu}_{kjg}$ and $\hat{\mu}_{kg}$, respectively. Because we have:

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

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

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

We remove the variance term $\left(\hat{\sigma}_{g2}^2\right)$ 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 Supplementary Note 1.

We also implemented the standard HVG selection procedure, which handles batch effects using the method in Seurat version 3 (ref. [33]), and document the implementation details in 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 and 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 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 equation (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 equation (16). We save the batch-correction factors for all genes, so that we can calculate batch-adjusted expression levels for any gene instantly in the future.

kBET acceptance rate. The kBET[34] acceptance rate measures whether 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 (k-NN) (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:

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

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 whether the neighbors of a cell have the same cell type as it does. If a method overcorrects the batch effects, it will have a low kSIM acceptance rate. We use the HNSW algorithm to find k-NNs (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:

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

where k and β are user-specified parameters.

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

k-NN graph construction. Pegasus uses the HNSW[35] algorithm with HNSW parameters $M=20$, $efC=200$ and $efS=200$ 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 first nearest neighbor. For any cell missing itself as the first nearest neighbor, Pegasus sets itself as the first nearest neighbor and picks the top 99 nearest neighbors returned by HNSW as the second to one-hundredth 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: a robust mode and a full-speed mode. In the robust mode, Pegasus runs the index-building process with only one thread and runs the neighbor-searching process with multiple threads. In the 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 $W_{N\times N}$ based on the top m principal components. This affinity matrix is also used in community-detection-based clustering algorithms. We construct W based on the top k nearest neighbors found by the HNSW algorithm. Let us define a cell x's neighborhood set $N(x)$ as the set consisting of x's second to the kth nearest neighbors and define d_i as the distance between cell x and its ith neighbor. In addition, we denote $\mathbf{x}$ as the vector containing the top m principal component coordinates of cell x. We then define the following locally scaled Gaussian kernel between any two cells x and y as:

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

where $\sigma_{\mathbf{x}}$ is x's local kernel width, defined as $\sigma_{\mathbf{x}} = \text{median}\{d_i | i = 2, \ldots, K\}$. To eliminate the effects of sampling density, we additionally define the following density-normalized kernel[14]:

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

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

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

The affinity matrix W is constructed using the density-normalized kernel:

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

We then calculate the Markov chain transition matrix P and the symmetric transition matrix Q based on the affinity matrix:

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

$$Q = D^{-\frac{1}{2}}WD^{-\frac{1}{2}}, \quad P = D^{-\frac{1}{2}}QD^{\frac{1}{2}} \tag{26}$$

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

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

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 Q using the implicitly restarted Lanczos method[56] (via the scipy.sparse.linalg.eigsh function). We also provide the alternative option to calculate the top n eigenvalues using the randomized singular value decomposition 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 $\{\Psi_t\}_{t\in\mathbb{N}}$ parameterized by timescale t:

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

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 symmetrical matrix Q instead.

Next, we define approximated DPT maps $\{\Psi'_t\}_{t\in\mathbb{N}\cup\{\infty\}}$ based on approximated diffusion maps:

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

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 smooths 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 P^t induces a graph with the following Laplacian and density matrices (Supplementary Note 1):

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

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

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

$S(t)$ increases as t increases, and reaches its maximum $\log[n-1]$ when $t\to\infty$. Because smaller eigenvalues of P (probably representing noise) decrease to 0 (and hence contribute a 1 to the entropy) much more rapidly than large eigenvalues (probably 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 Fig. 4b). We pick the timescale t as the knee point of this von Neumann entropy curve.

Let us denote $p_t = \begin{bmatrix} t \\ S(t) \end{bmatrix}$ as the point for timescale t in the curve. We find the knee point by scanning all ts (where t is an integer) between 1 and t_{max}, and pick the t that is furthest from $p_{t_{max}} - p_1$—the segment connecting the two endpoints of the curve. t_{max} is a user-provided parameter set to $t_{max} = 5{,}000$ by default. We calculate the distance between point p_t and the segment $p_{t_{max}} - p_1$ as follows:

$$\text{dis} = \frac{\|(p_t - p_1) \times (p_{t_{max}} - p_1)\|}{\|p_{t_{max}} - p_1\|} \quad (33)$$

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

$$D'_t(\mathbf{r}, \mathbf{x}) = \Psi'_t(\mathbf{r}) - \Psi'_t(\mathbf{x}) \quad (34)$$

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 W. In the graph, vertex set V contains all cells, and an edge $(u,v) \in E$ only if $W_{u,v} > 0$. The weight of the edge is calculated as:

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

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

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

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

$$m = \sum_{(u,v) \in E} w(u,v), \quad e_c = \sum_{u \in c, v \in c, (u,v) \in E} w(u,v) \quad \text{and} \quad K_c = \sum_{u \in c} \sum_{(u,v) \in E} w(u,v) \quad (37)$$

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 Q; and (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 V. Traag (https://github.com/vtraag/louvain-igraph). Note that the latest release of the louvain-igraph package (version 0.6.1) 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; and (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 probably 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 DPT component space, using a two-level clustering strategy. We first partition the cells into k_1 clusters using the scikit-learn 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 DPT components are not calculated, Pegasus will apply the k-means algorithm on the principal components instead.

t-SNE, UMAP and FLE. Pegasus calculates a t-SNE using the Multicore-TSNE package implemented by D. 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 = 1,000, n_iter = 1,000 and n_iter_early_exag = 250.

Pegasus calculates a FIt-SNE, which is a fast approximation of t-SNE embedding, using the pyFIt-SNE package from the Kluger laboratory (https://github.com/KlugerLab/pyFIt-SNE). Pegasus uses the following FIt-SNE parameters by default: perplexity = 30, early_exaggeration = 12, learning_rate = 1,000, max_iter = 1,000, 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 L. Mclnnes (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 on the Gephi ForceAtlas2 (ref. [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 $n_\delta = 5{,}000$ 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:

$$p_i \propto \frac{1}{d_i^\alpha}, \quad (38)$$

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 one input layer and four hidden layers. The input layer connects with the top m principal components for each cell and the four hidden layers contain 100, 75, 50 and 25 rectified linear units, respectively. We train the network using scikit-learn's MLPRegressor with a stochastic gradient decent solver, an adaptive learning rate and an L2 penalty parameter of 0.1. We additionally scale the inputs and outputs of the neural network, such that the maximum standard deviations 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, $n_\delta = 1{,}500$.

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 at 5% using the Benjamini–Hochberg procedure[58]. Pegasus can optionally calculate the AUROC for each gene by considering the binary classification problem that uses the gene as the only feature to predict whether a cell is within or outside a cluster. Pegasus performs these analyses in parallel across genes to speed up the calculation process and outputs results for all genes in a spreadsheet. 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 extracts 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 three groups using the k-means algorithm, and assigns each of the three groups as strongly upregulated, weakly upregulated or downregulated 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.readthedocs.io/en/0.14.0/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 upregulated, its impact value is 0. Otherwise, if the fold-change (FC) in the percentage of cells expressing this marker (within versus outside the cluster) satisfies $FC \geq 1.5$, 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 upregulated, its impact value is 0. If it is neither upregulated nor downregulated, 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 version 0.15.0 and benchmarked it with SCANPY version 1.4.4.post1 and Seurat version 3.1.0 on key analysis tasks (Supplementary Tables 4–6) using a high-performance local server (cloud-based times as part of a Cumulus workflow are shown in Table 1). The server has 28 CPU threads (one Intel Xeon E5–2660 v4 processor; 14 cores; 2.00 GHz; 35 MB cache) and 256 GB 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 the cloud with running the Cell Ranger + Seurat/SCANPY pipeline (on a 32-CPU-thread, 120-GB Google Cloud virtual machine) on the bone marrow dataset. We used Cell Ranger version 2.2.0 for this benchmark. The virtual machine for running Seurat (version 3.1.0) and SCANPY (version 1.4.4.post1) was created on Google Compute Engine zone us-west1-d—the same zone on which Cumulus ran its analysis step.

Bone marrow dataset preprocessing. We preprocessed the bone marrow dataset 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 the 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, 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 HVGs selected using the two procedures with a list of immune genes curated by the ImmPort[28] team (Extended Data Fig. 1b) from https://www.immport.org/shared/geneData/GOappend1.xls, which contains 1,534 genes (with duplicates) annotated with immune-related Gene Ontology 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:

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

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 version 3 using a subset of data from the bone marrow dataset, consisting of the first 10x Genomics channel from each of the eight 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 Fig. 2b).

The clustering results showed one particular donor-specific effect (Extended Data Fig. 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 five 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 five adjacent T cell clusters.

To generate batch-corrected results, we applied the same preprocessing step to the 34,654-cell dataset, selected the 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- and MNN-corrected expression levels. With Seurat version 3 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' *k*-NN graph and kept the other analyses the same.

We used kBET and kSIM acceptance rates on UMAP 2D coordinates. For the kBET acceptance rate, we set $k = 25$ and $\alpha = 0.05$. For the 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 principal components produced by Pegasus and sought for the 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*-NNs 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 performed *k*-NN graph construction as described in the 'HVG selection' section. We then generated diffusion pseudotime maps (with the parameters noted in Fig. 2b and Extended Data Fig. 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 the new HVG procedure).

Spectral community detection algorithms. We preprocessed the bone marrow dataset, selected HVGs using the new procedure, corrected batch effects, ran PCA, performed *k*-NN graph construction and generated a 2D visualization using FIt-SNE, as described in the 'HVG selection' section. We calculated a diffusion pseudotime map with default parameters. We then generated different cluster 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, selected HVGs using the new procedure, corrected batch effects, ran PCA, performed *k*-NN graph construction, detected communities using the Louvain algorithm, performed differential expression analysis and performed marker-based cell type annotation as described in the 'HVG selection' section. We calculated a diffusion pseudotime map using default parameters. We then ran Net-tSNE and compared it with *t*-SNE and FIt-SNE; ran Net-UMAP and compared it with UMAP;and ran Net-FLE and compared it with FLE.

Benchmarking Pegasus, SCANPY and Seurat on the full bone marrow dataset. We benchmarked Pegasus, SCANPY and Seurat on ten 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 was Seurat, which does not accept a pre-computed affinity matrix as input for clustering. Thus, we provided Seurat with pre-computed principal components 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 10× channel a separate batch, which resulted in 63 batches in total. We used the integration method of Seurat version 3 (ref. [33]), BBKNN[32] and the 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 they were 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; thus, we included the *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 version 3 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 contain 1,306,127 cell barcodes. We preprocessed this dataset 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 the total time by multiplying by a factor of ten. 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 5,000, 10,000, 25,000, 50,000, 100,000 and 200,000 cells. Together with the full dataset, this resulted in seven datasets of various sizes. We then benchmarked Pegasus on the seven datasets and recorded the execution time of each Pegasus component, including HVG selection (hvg), batch correction, PCA, *k*-NN 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—the cloud setting (28 threads) and the laptop setting (eight threads)—to test whether 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 that these improvements are faster in both cloud and laptop settings once the number of cells reaches 25,000. In addition, these improvements are as efficient as their counterparts for small datasets (that is, 5,000 and 10,000). The Pegasus commands can be found in Supplementary Note 2.

Benchmarking Pegasus, SCANPY and Seurat version 3 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 dataset 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 for 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 version 3 peak memory on the 5K PBMC and bone marrow datasets. We measured the peak memory usage of Pegasus, SCANPY and Seurat version 3 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 for all three tools. In addition, for all three tools, we did not execute the tasks that Seurat does not provide (Supplementary Table 4), to make sure the measured peak memory usages were comparable. We used eight 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.

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 much less expensive. By default, Cumulus allows up to two tries using preemptible instances before switching to non-kicked-off instances. Cumulus execution time is readout from Terra execution logs. To estimate the execution time of mkfastq and count steps on a 32-CPU-thread virtual machine (Table 1), we only sum over Terra-reported Docker running times of successful mkfastq and count runs, respectively. The analysis times for Seurat and SCANPY are estimated by running each tool on the same 32-CPU-thread, 120-GB memory Google Cloud virtual machine instance. Since both Seurat and SCANPY do not have functions to aggregate 10× 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 (Fig. 2e), we reduced the number of batches from 63 to eight (one per donor) and the number of principal components used from 30 (default) to 20. See Supplementary Note 2 for details on how the Seurat, SCANPY and Cumulus analysis steps were run. The total computational costs were reported by Terra and we calculated the average cost per sample by dividing by 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 two 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 (that is, total cost/total runtime × one count job runtime) (Supplementary Data 3). We plotted execution time and amortized total cost against channels in Extended Data Fig. 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 the cloud and a local server (96 CPU threads; 1 TB memory) to generate gene-count matrices for one 10× Genomics version 3 dataset and one SMART-seq2 dataset and confirmed that the matrices were identical. Specifically, we chose a 5K PBMC dataset of a healthy donor from the 10x Genomics website (https://support.10xgenomics.com/single-cell-gene-expression/datasets/3.0.2/5k_pbmc_v3) and ran Cell Ranger version 3.1.0 with Cell Ranger human reference GRCh38 version 3.0.0. For SMART-seq2, we picked four 50-base-pair single-end and four 100-base-pair paired-end mouse embryonic stem cells sequenced by BGISEQ-500 (ref. [60]) and ran RSEM + HISAT2 with a transcript reference extracted from the 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 into one FASTQ file for Optimus, Alevin, Kallisto-BUStools and STARsolo. We then asked Cumulus to run Cell Ranger (version 3.1.0), Optimus (version 1.4.0+; forked from the GitHub master branch), Alevin (version 1.1), Kallisto-BUStools (version 0.24.4) and STARsolo (version 2.7.3a) 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 summarize 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 the 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.

Reporting Summary. Further information on research 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 dataset is available at https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.3.0/1M_neurons. The 5K PBMC dataset is available at https://support.10xgenomics.com/single-cell-gene-expression/datasets/3.0.2/5k_pbmc_v3. The BGISEQ SMART-Seq2 data are 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 were used.

Code availability

Cumulus code consists of four components: the Pegasus and scPlot python packages; the Cumulus WDL workflows and Docker files; 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 the workflows using the Workflow Description Language (WDL; https://github.com/openwdl/wdl) and encapsulated all software packages into Docker images using Docker files. Cumulus WDL and Docker files are available at https://github.com/klarman-cell-observatory/cumulus. The source code used to generate feature-count matrices, generate_count_matrix_ADTs, is 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 deposited 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, Docker files and Cirrocumulus are licensed under a BSD three-clause license. In addition, we documented licenses for Cumulus dependencies in Supplementary Data 5. Due to third-party licensing requirements, we can only provide CellRanger dockers without bcl2fastq2 and users can build

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

References

46. Guo, R., Zhao, Y., Zou, Q., Fang, X. & Peng, S. Bioinformatics applications on Apache Spark. *Gigascience* **7**, giy098 (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**, 78 (2018).
48. Li, B. & Dewey, C. N. RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. *BMC Bioinform.* **12**, 323 (2011).
49. Kim, D., Paggi, J. M., Park, C., Bennett, C. & Salzberg, S. L. Graph-based genome alignment and genotyping with HISAT2 and HISAT-genotype. *Nat. Biotechnol.* **37**, 907–915 (2019).
50. Langmead, B. & Salzberg, S. L. Fast gapped-read alignment with Bowtie 2. *Nat. Methods* **9**, 357–359 (2012).
51. Dobin, A. et al. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics* **29**, 15–21 (2013).
52. Dixit, A. Correcting chimeric crosstalk in single cell RNA-seq experiments. Preprint at *bioRxiv* https://doi.org/10.1101/093237 (2016).
53. Cleveland, W. S., Grosse, E. & Shyu, W. M. in *Statistical Models in S* Ch. 8 (1992).
54. Halko, N., Martinsson, P. G. & Tropp, J. A. 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, D. C. 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, N. X., 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 (Association for Computing Machinery, 2009).
60. Natarajan, K. N. et al. Comparative analysis of sequencing technologies for single-cell transcriptomics. *Genome Biol.* **20**, 70 (2019).

Acknowledgements

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 an open-source software, 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 contributions in generating the Census of Immune Cells dataset 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 the 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.), as well as the Human Tumor Atlas Pilot Project scientific team at Leidos Biomedical Research, Frederick National Laboratory for Cancer Research and NCI.

Author contributions

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

Competing interests

A.R. is a founder of and equity holder in Celsius Therapeutics, a Scientific Advisory Board member of Thermo Fisher Scientific, Neogene Therapeutics, Syros Pharamceuticals and Asimov, and an equity holder in Immunitas. N.H. is a founder and Scientific Advisory Board member of Neon Therapeutics. A.R. is a co-inventor on patent applications filed by the Broad Institute to inventions relating to single-cell genomics.

Additional information

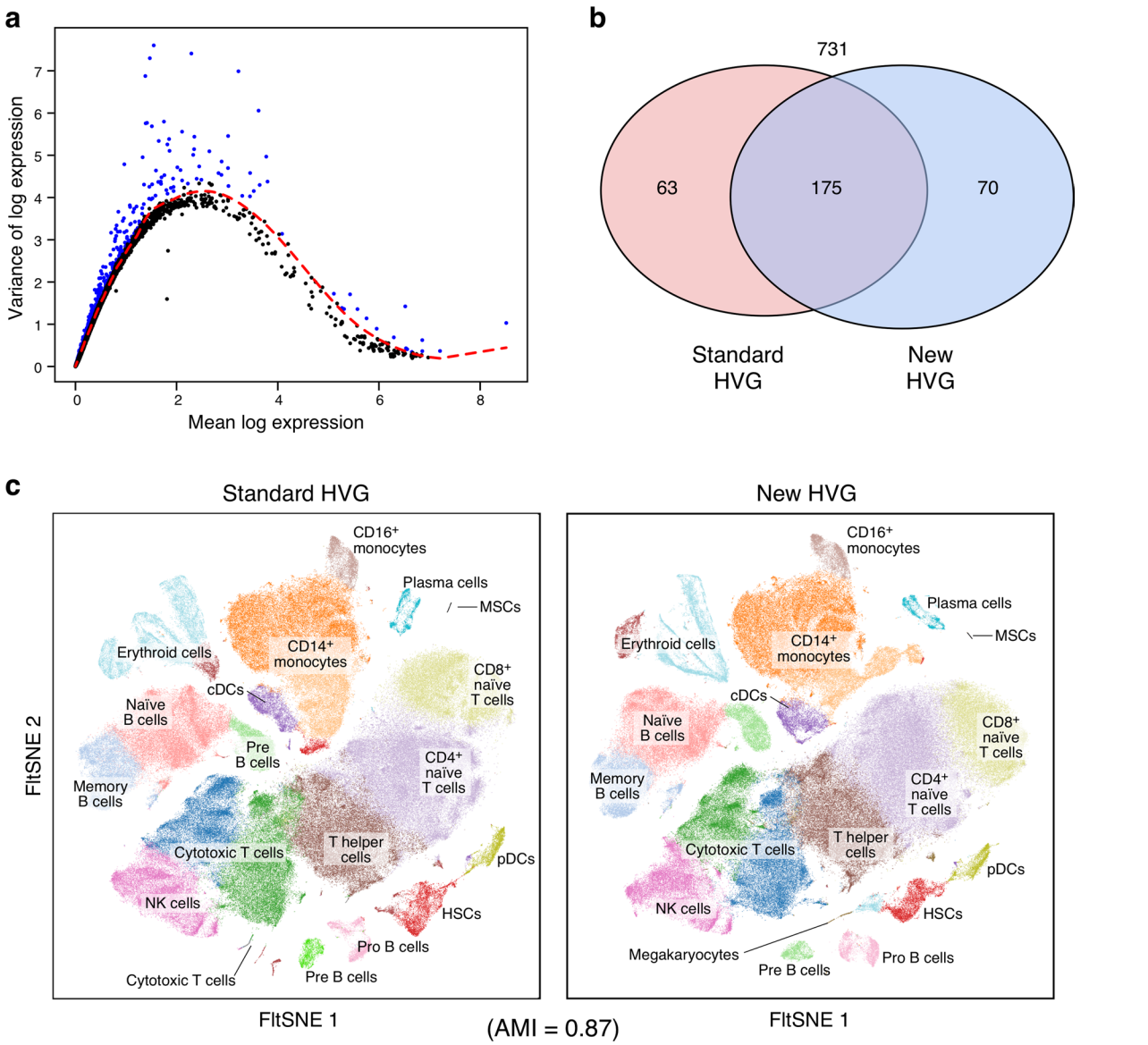
Extended data is available for this paper at https://doi.org/10.1038/s41592-020-0905-x.

Supplementary information is available for this paper at https://doi.org/10.1038/s41592-020-0905-x.

Correspondence and requests for materials should be addressed to B.L., O.R.-R. or A.R.

Peer review information Lin Tang was the primary editor on this article and managed its editorial process and peer review in collaboration with the rest of the editorial team.

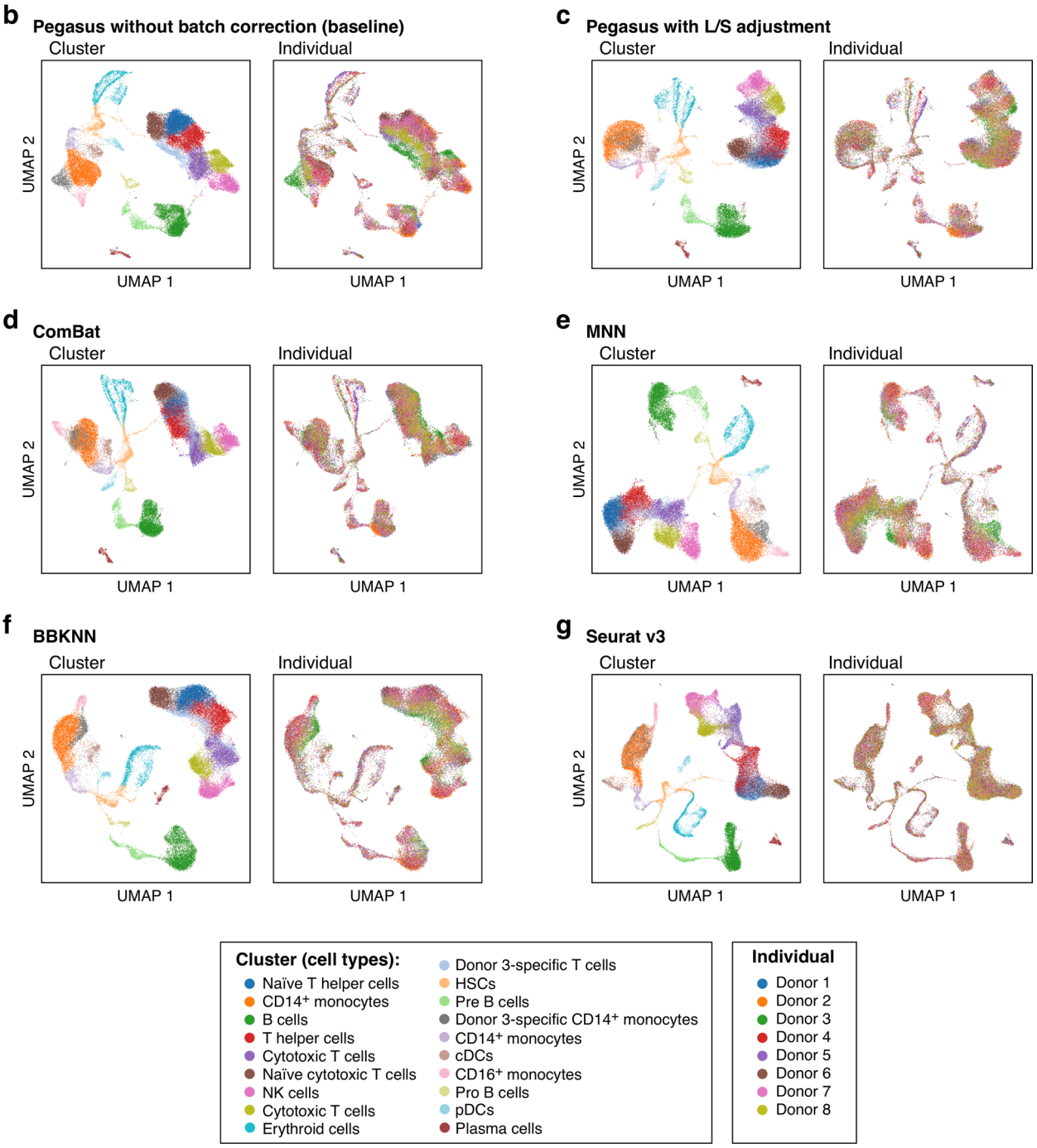
Reprints and permissions information is available at www.nature.com/reprints.



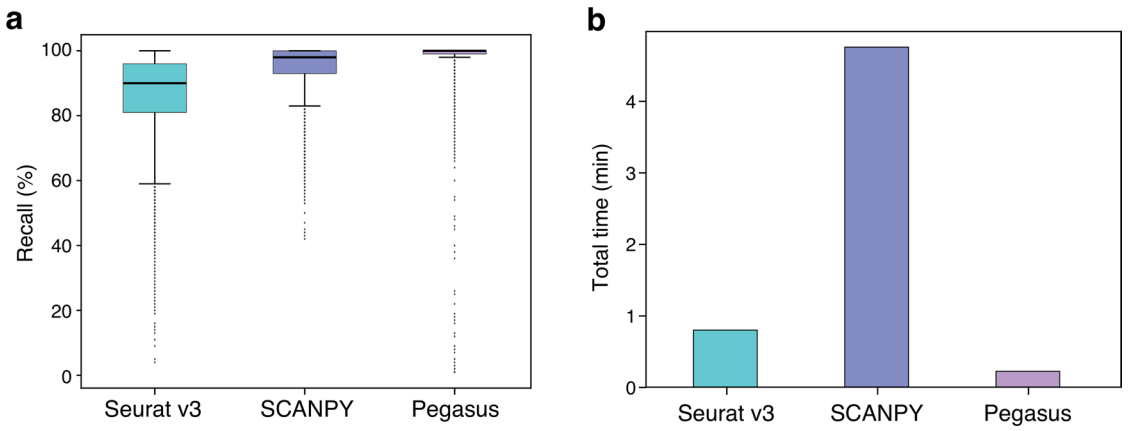
Extended Data Fig. 1 | The new HVG selection procedure provides excellent quality vs. the standard procedure. a, New HVG selection procedure (n = 16,613 robust genes). Variance (y axis) vs. mean (x axis) of log expression. Red: fitted LOESS curve. HVGs (blue) are defined as the genes above the LOESS 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.

a

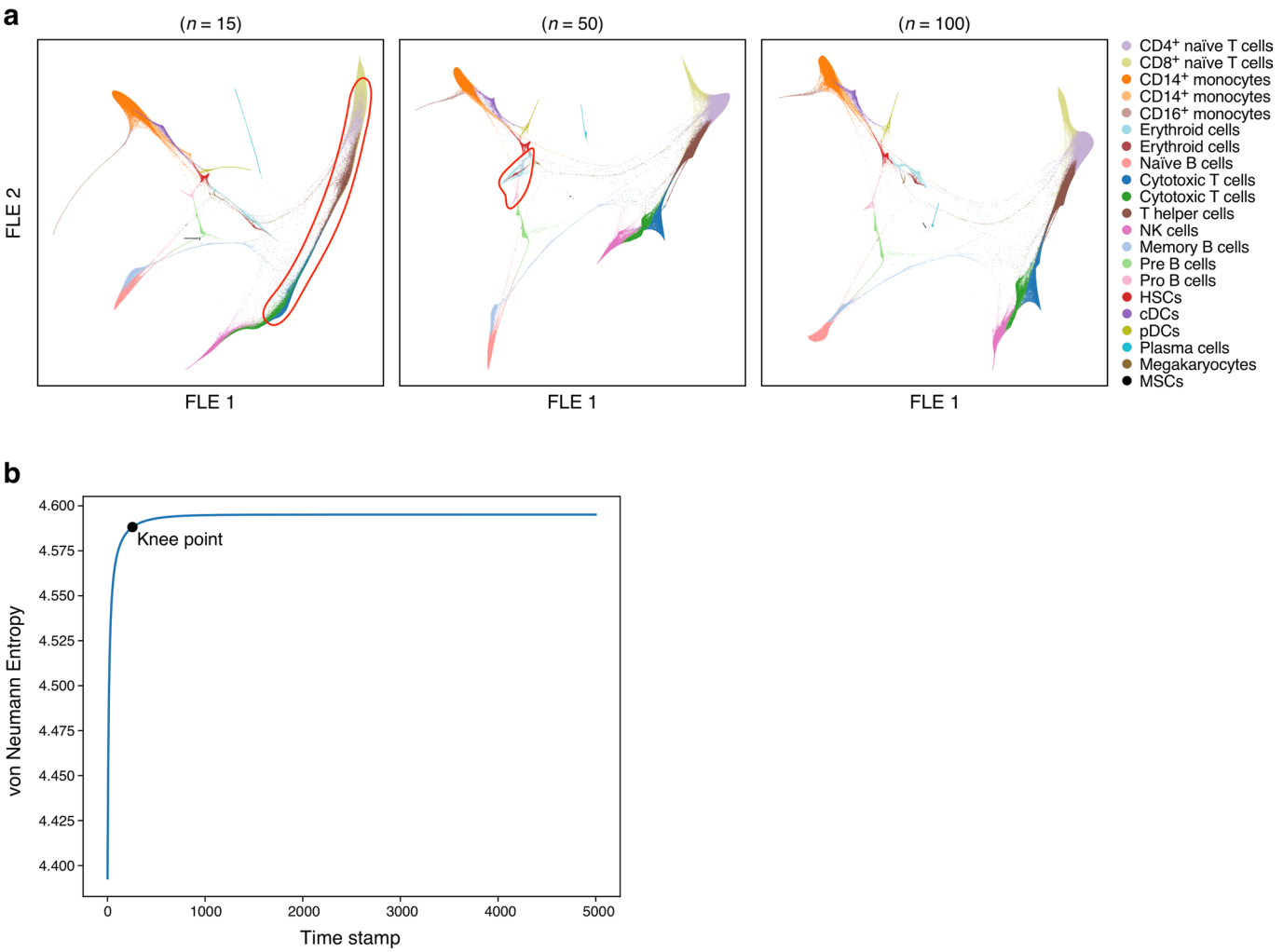
	Pegasus	ComBat	MNN	BBKNN	Seurat v3
Batch correction time:	2 s	20 s	33 min, 55 s	10 s	8 min, 19 s



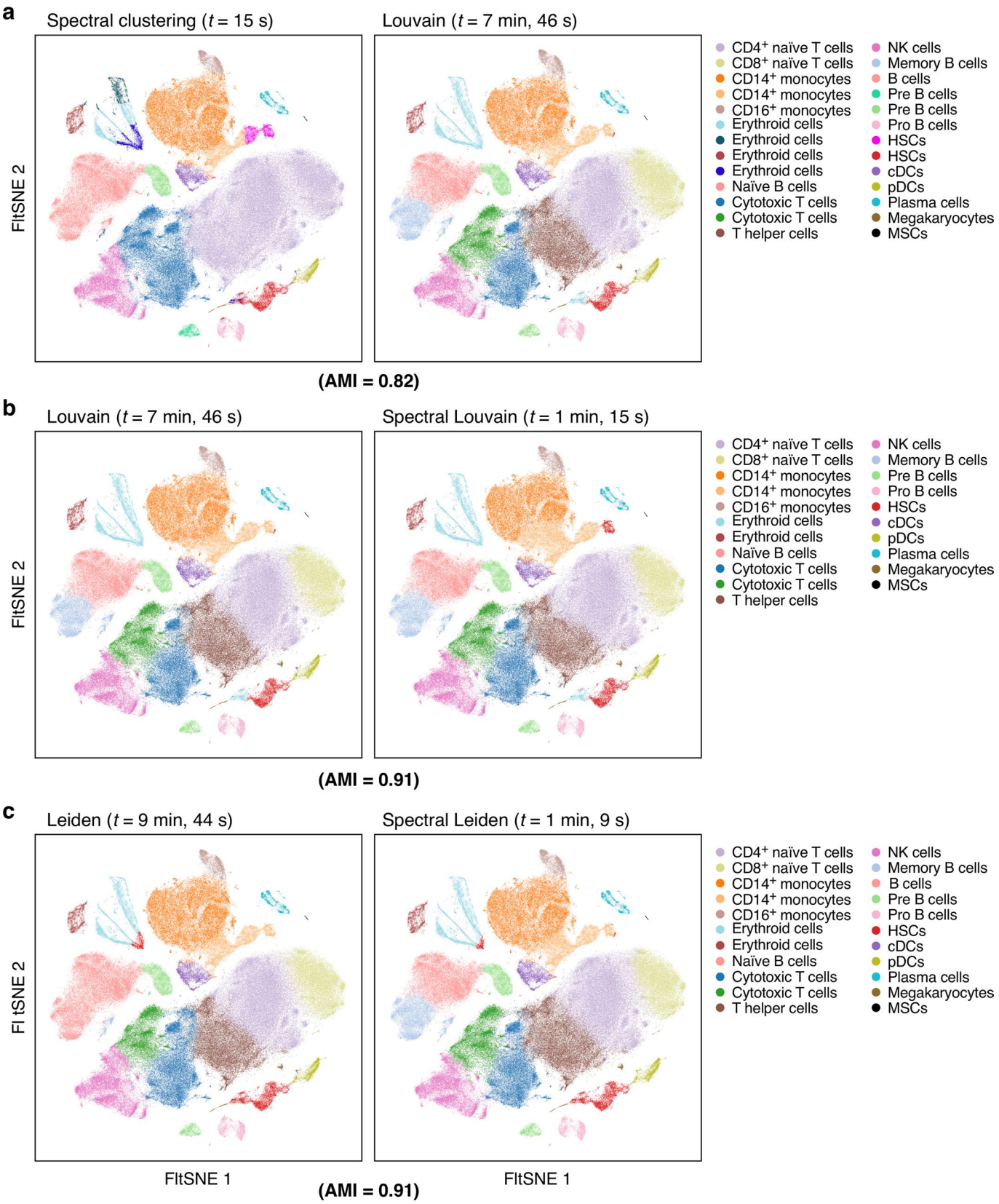
Extended Data Fig. 2 | Benchmarking of batch correction methods on 34,654 bone marrow cells. a, Execution time of each method. **b–g**, UMAP visualizations of the bone marrow cells (n = 34,654) colored by either cell type annotation (left) or donor identity (right) without batch correction (**b**, baseline),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



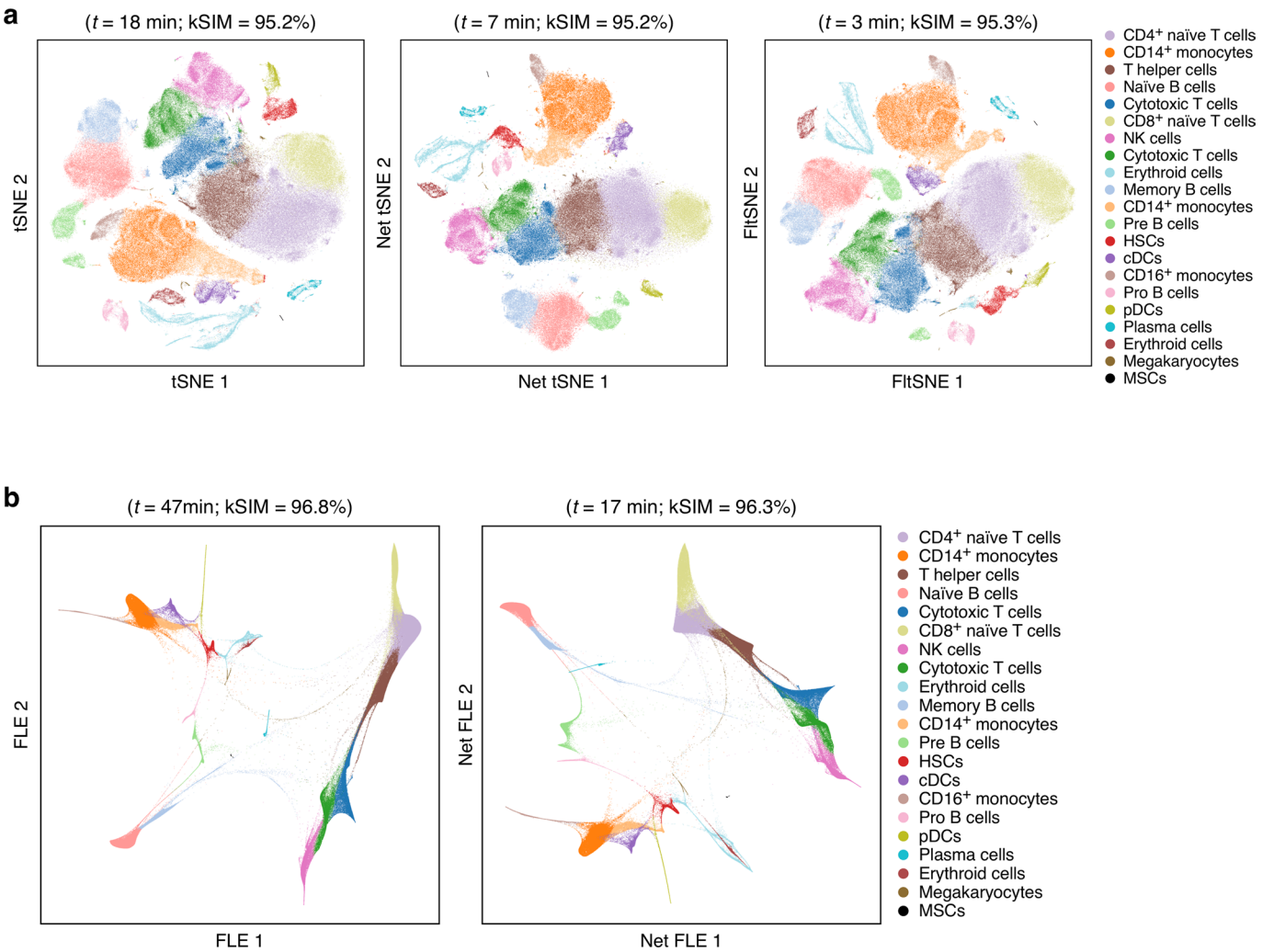
Extended Data Fig. 3 | Benchmarking of approximate nearest neighbor finding methods on the bone marrow dataset (n = 274,182 cells). Accuracy (**a**, y axis, % recall, Methods) and speed (**b**, y axis, minutes) of each of three methods. Boxplot (a): Line: median; box boundaries: lower and upper quartiles; whiskers: 1.5 interquartile range (IQR) below and above the low and high quartile, respectively.



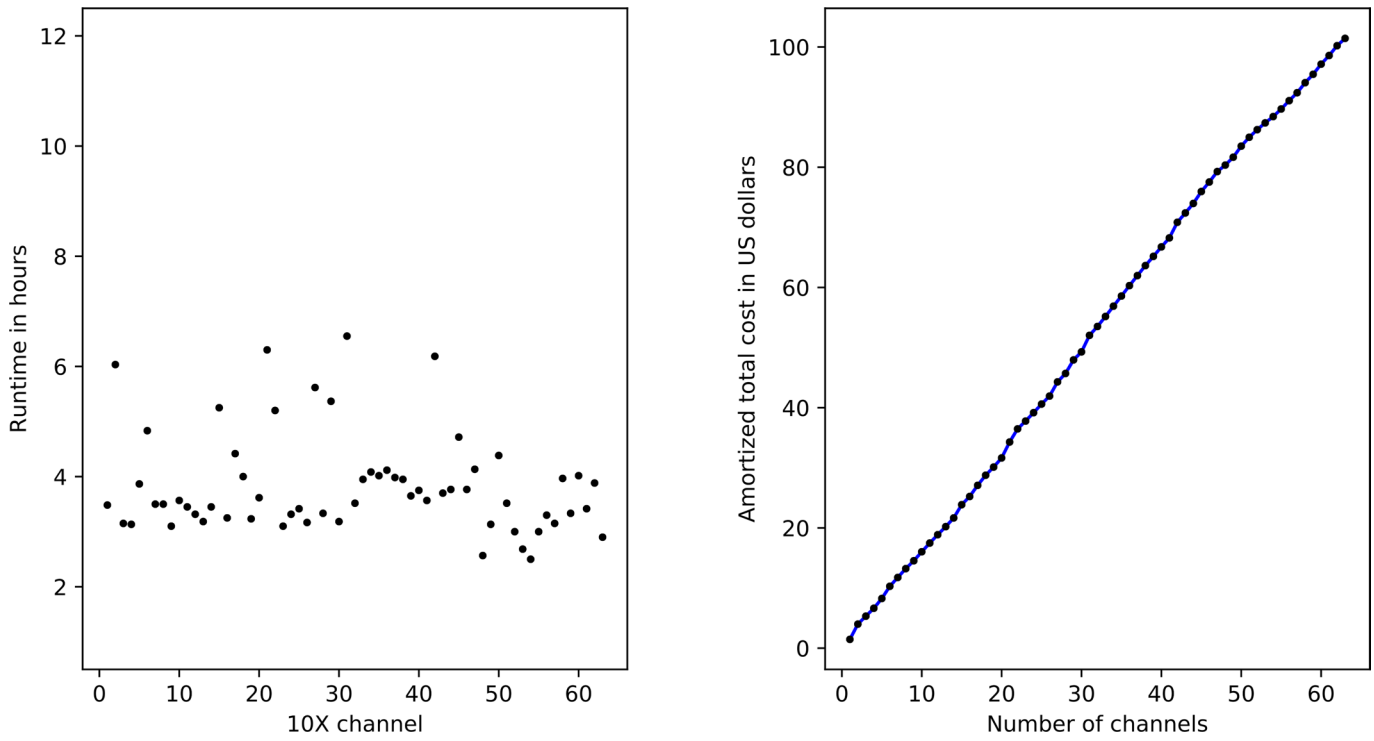
Extended Data Fig. 4 | Adjusting diffusion pseudotime map parameters for visualization of pseudotemporal trajectories. a, Using a large number of diffusion pseudotime components yields a developmental trajectory that enhances separation of trajectories of different cell populations (n = 274,182 bone 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.



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.



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.



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.

nature research

Corresponding author(s): Aviv Regev (aregev@broadinstitute.org)

Last updated by author(s): Apr 29, 2020

Reporting Summary

Nature Research wishes to improve the reproducibility of the work that we publish. This form provides structure for consistency and transparency in reporting. For further information on Nature Research policies, see Authors & Referees and the Editorial Policy Checklist.

Statistics

For all statistical analyses, confirm that the following items are present in the figure legend, table legend, main text, or Methods section.

n/a	Confirmed	
☐	☒	The exact sample size (*n*) for each experimental group/condition, given as a discrete number and unit of measurement
☐	☒	A statement on whether measurements were taken from distinct samples or whether the same sample was measured repeatedly
☐	☒	The statistical test(s) used AND whether they are one- or two-sided *Only common tests should be described solely by name; describe more complex techniques in the Methods section.*
☐	☒	A description of all covariates tested
☐	☒	A description of any assumptions or corrections, such as tests of normality and adjustment for multiple comparisons
☐	☒	A full description of the statistical parameters including central tendency (e.g. means) or other basic estimates (e.g. regression coefficient) AND variation (e.g. standard deviation) or associated estimates of uncertainty (e.g. confidence intervals)
☒	☐	For null hypothesis testing, the test statistic (e.g. *F*, *t*, *r*) with confidence intervals, effect sizes, degrees of freedom and *P* value noted *Give P values as exact values whenever suitable.*
☒	☐	For Bayesian analysis, information on the choice of priors and Markov chain Monte Carlo settings
☒	☐	For hierarchical and complex designs, identification of the appropriate level for tests and full reporting of outcomes
☒	☐	Estimates of effect sizes (e.g. Cohen's *d*, Pearson's *r*), indicating how they were calculated

Our web collection on statistics for biologists contains articles on many of the points above.

Software and code

Policy information about availability of computer code

Data collection	Not relevant.
Data analysis	1) Cumulus v0.15.0, which consists of Dockerfiles and WDL workflows, Pegasus analysis package and Cirrocumulus visualizer, is developed as the main contribution of this paper. 2) Pegasus v0.15.0, SCANPY v1.4.4.post1 and Seurat v3.1.0 are used for benchmarking purpose. 3) 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 are benchmarked with respect to cloud execution speed for generating gene-count matrices. 4) Cirrocumulus v1.0.0 is used in Supplementary Video 3 for demonstration.

For manuscripts utilizing custom algorithms or software that are central to the research but not yet described in published literature, software must be made available to editors/reviewers. We strongly encourage code deposition in a community repository (e.g. GitHub). See the Nature Research guidelines for submitting code & software for further information.

Data

Policy information about availability of data

All manuscripts must include a data availability statement. This statement should provide the following information, where applicable:

- Accession codes, unique identifiers, or web links for publicly available datasets
- A list of figures that have associated raw data
- A description of any restrictions on 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.

Field-specific reporting

Please select the one below that is the best fit for your research. If you are not sure, read the appropriate sections before making your selection.

☒ Life sciences ☐ Behavioural & social sciences ☐ Ecological, evolutionary & environmental sciences

For a reference copy of the document with all sections, see nature.com/documents/nr-reporting-summary-flat.pdf

Life sciences study design

All studies must disclose on these points even when the disclosure is negative.

Sample size	We have benchmarked Pegasus and alternative tools on three publicly available data sets: human bone marrow, mouse brain and human PBMC. We additionally benchmarked Cumulus and alternatives on the human bone marrow data set. The human bone marrow dataset has 274,182 cells after filtration. The mouse brain dataset has 1,286,072 cells after filtration. The human PBMC dataset has 4,515 cells after filtration. There is no sample size calculation involved. Because all three datasets are publicly available datasets produced before this paper, there is no way that we can control their sample size (i.e. number of single cells). However, we have good reasons to choose these 3 datasets. We generated the bone marrow dataset as part of the Human Immune Cell Atlas effort and thus have a better understanding of this data. As a result, the bone marrow dataset is ideal for us to test/validate implementation/algorithmic improvements Pegasus has made. The mouse brain and human PBMC datasets represent large (~1.3M cells) and small (~ 4.5K cells) datasets and serve as good test cases showing that Pegasus works well for datasets at a variety of scales.
Data exclusions	Low quality single cells were filtered out according to per-established criteria, which are clearly documented in the online Methods.
Replication	The bone marrow dataset was collected from 8 donors with equal gender balance, consisting of 63 10x channels (V2 chemistry). The mouse brain dataset was collected from 2 embryonic day 18 mice, consisting of 133 10x channels (V2 chemistry). The human PBMC dataset was collected from one donor, consisting of 1 10x channel (V3 chemistry). The replication information has been clearly documented at the websites providing these publicly available datasets.
Randomization	We do not have experimental groups.
Blinding	We do not have experimental groups.

Reporting for specific materials, systems and methods

We require information from authors about some types of materials, experimental systems and methods used in many studies. Here, indicate whether each material, system or method listed is relevant to your study. If you are not sure if a list item applies to your research, read the appropriate section before selecting a response.

Materials & experimental systems

n/a	Involved in the study
☒	☐ Antibodies
☒	☐ Eukaryotic cell lines
☒	☐ Palaeontology
☒	☐ Animals and other organisms
☒	☐ Human research participants
☒	☐ Clinical data

Methods

n/a	Involved in the study
☒	☐ ChIP-seq
☒	☐ Flow cytometry
☒	☐ MRI-based neuroimaging