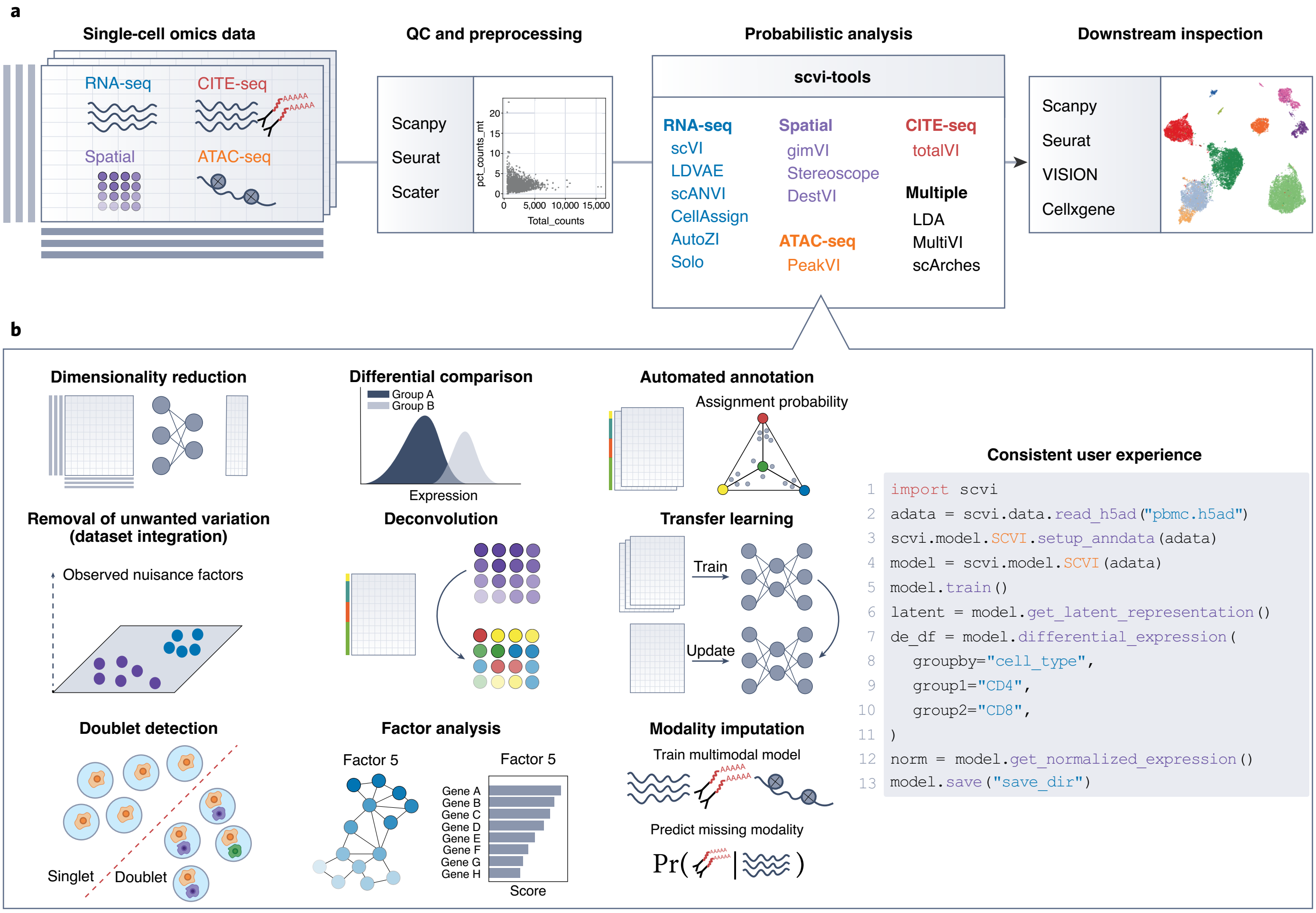
A Python library for probabilistic analysis of single-cell omics data

To the Editor — Methods for analyzing single-cell data[1–4] perform a core set of computational tasks. These tasks include dimensionality reduction, cell clustering, cell-state annotation, removal of unwanted variation, analysis of differential expression, identification of spatial patterns of gene expression, and joint analysis of multi-modal omics data. Many of these methods rely on likelihood-based models to represent variation in the data; we refer to these as 'probabilistic models'. Probabilistic models provide principled ways to capture uncertainty in biological systems and are convenient for decomposing the many sources of variation that give rise to omics data[5].

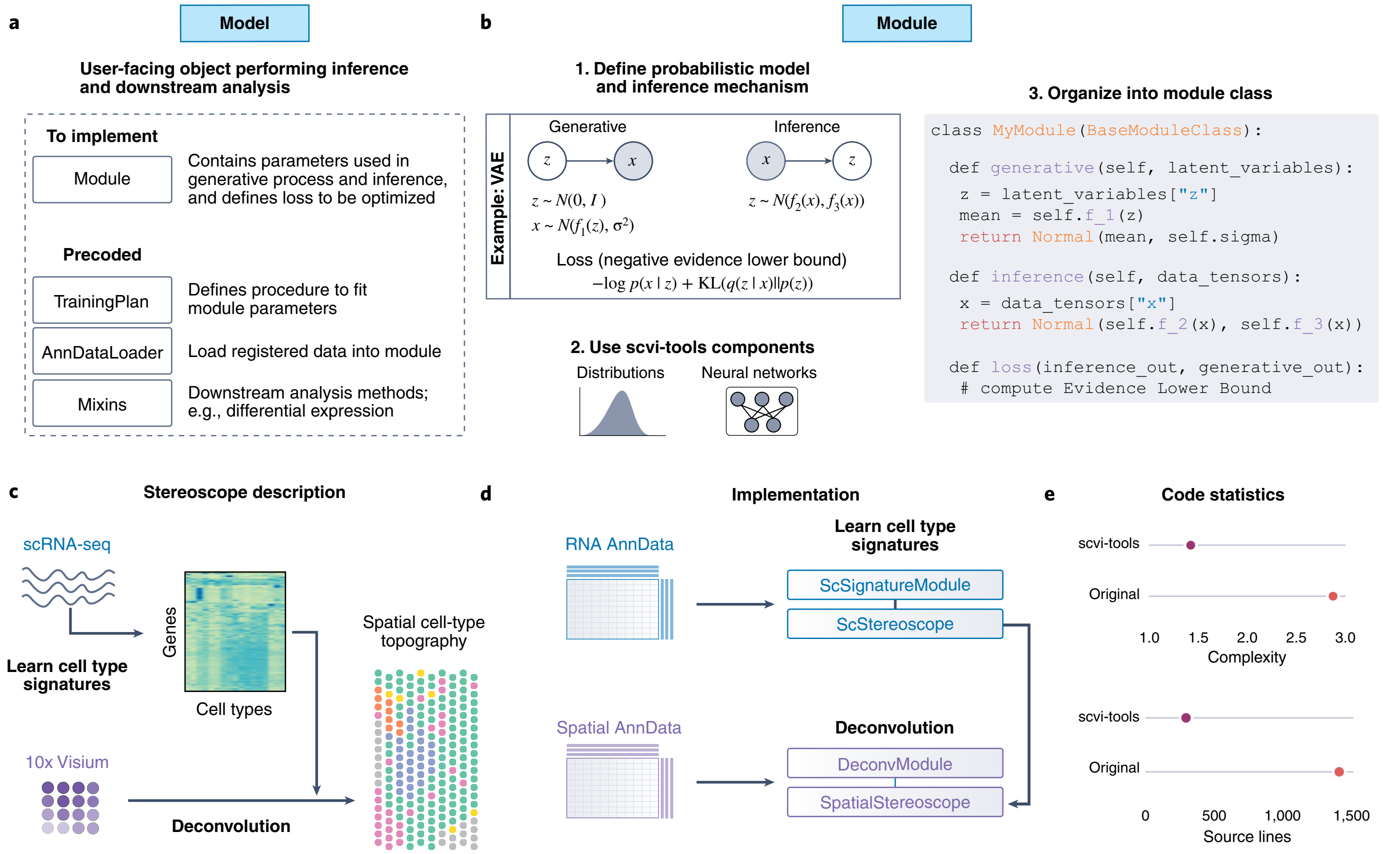
Despite the appeal of probabilistic models, several obstacles impede their community-wide adoption. The first obstacle, coming from the perspective of the end user, relates to the difficulty of implementing and running such models. Because probabilistic models are often implemented using Python machine-learning libraries, users are often required to interact with interfaces and objects that are lower level in nature than those used in popular environments for single-cell data analysis like Bioconductor[6], Seurat[7] or Scanpy[8].

A second obstacle relates to the development of new probabilistic models. From the perspective of developers, there are many necessary routines to implement in support of a probabilistic



```
import scvi
adata = scvi.data.read_h5ad("pbmc.h5ad")
scvi.model.SCVI.setup_anndata(adata)
model = scvi.model.SCVI(adata)
model.train()
latent = model.get_latent_representation()
de_df = model.differential_expression(
    groupby="cell_type",
    group1="CD4",
    group2="CD8",
)
norm = model.get_normalized_expression()
model.save("save_dir")
```

Fig. 1 | User perspective of scvi-tools. a, Overview of single-cell omics analysis pipeline with scvi-tools. Datasets may contain multiple layers of omic information, along with metadata at the cell and feature levels. QC and preprocessing are done with popular packages like Scanpy, Seurat and Scater. Subsequently, datasets can be analyzed with scvi-tools, which contains implementations of probabilistic models that offer a range of capabilities for various omics. Finally, results are further investigated or visualized, typically through a nearest neighbors graph, and through environments like VISION or cellxgene or by directing back to Scanpy or Seurat. **b**, Left, overview of the functionality of models implemented in scvi-tools covers core single-cell analysis tasks. Right, each model has a simple and consistent user interface; the code snippet shown applies scVI to a dataset read from a h5ad file and then performs dimensionality reduction and differential expression.


a
Model
User-facing object performing inference and downstream analysis
To implement
Module
Contains parameters used in generative process and inference, and defines loss to be optimized
Precoded
TrainingPlan
Defines procedure to fit module parameters
AnnDataLoader
Load registered data into module
Mixins
Downstream analysis methods; e.g., differential expression
b
Module
1. Define probabilistic model and inference mechanism
Example: VAE
Generative
Inference
z
x
$z \sim N(0, I)$
$x \sim N(f_1(z), \sigma^2)$
$z \sim N(f_2(x), f_3(x))$
Loss (negative evidence lower bound)
$-\log p(x \mid z) + \mathrm{KL}(q(z \mid x) \| p(z))$
2. Use scvi-tools components
Distributions
Neural networks
3. Organize into module class
c
Stereoscope description
scRNA-seq
Genes
Cell types
Learn cell type signatures
Spatial cell-type topography
10x Visium
Deconvolution
d
Implementation
RNA AnnData
Learn cell type signatures
ScSignatureModule
ScStereoscope
Spatial AnnData
Deconvolution
DeconvModule
SpatialStereoscope
e
Code statistics
scvi-tools
Original
1.0
1.5
2.0
2.5
3.0
Complexity
scvi-tools
Original
0
500
1,000
1,500
Source lines

```
class MyModule(BaseModuleClass):

  def generative(self, latent_variables):
    z = latent_variables["z"]
    mean = self.f_1(z)
    return Normal(mean, self.sigma)

  def inference(self, data_tensors):
    x = data_tensors["x"]
    return Normal(self.f_2(x), self.f_3(x))

  def loss(inference_out, generative_out):
    # compute Evidence Lower Bound
```

Fig. 2 | **The scvi-tools API for developers and reimplementation of Stereoscope.** **a**, For every probabilistic method implemented in scvi-tools, users interact with a high-level 'model' object. The model relies on several lower level components for training a model and analyzing data. The 'module', which must be implemented, systematically encapsulates the probabilistic specification of the method. The rest of the lower level components rely on precoded objects in scvi-tools, such as AnnDataLoader for loading data from AnnData objects, TrainingPlan for updating the parameters of the module, and Mixin classes for downstream analyses. **b**, The creation of a new module in scvi-tools involves three key steps. First, the generative model and inference procedure are mathematically specified. Second, users may either choose from our wide range of precoded neural network architectures and distributions or implement their own with PyTorch. Finally, those elements are combined together and organized into a class that inherits from the abstract class BaseModuleClass (note: presentation is pseudocode). The generative method maps latent variables to the data-generating distribution. The inference method maps input data to the variational distribution (specific to variational inference). The loss method specifies the objective function for the training procedure, here the evidence lower bound (and specifically depicted for a variational autoencoder (VAE)). **c**, Overview of the Stereoscope method. Stereoscope takes as input a spatial transcriptomics dataset, as well as a single-cell RNA sequencing dataset, and outputs the proportion of cell types in every spot. **d**, Short description of the steps required to reimplement Stereoscope into the codebase. For each of the two models of Stereoscope, we created a module class as well as a model class. **e**, Average cyclomatic code complexity and total number of source code lines for each of scvi-tools implementation and the original implementation.

model, including data handling, tensor computations, training routines that handle device management (for example, GPU (graphic processing unit) computing), and the underlying optimization, sampling and numerical procedures. Although higher level machine-learning packages that automate some of these routines (for example, PyTorch Lightning[9] or Keras[10]) are becoming popular, they do not work seamlessly with single-cell omics data.

To address these limitations, we present scvi-tools (https://scvi-tools.org/), a Python library for deep probabilistic analysis of single-cell omics data. From the end user's perspective (Supplementary Note 1), scvi-tools offers standardized access to methods for many single-cell data analysis tasks, such as integration of single-cell RNA sequencing (scRNA-seq) data (scVI[11] or scArches[12]), annotation of single-cell profiles (CellAssign[13] or scANVI[14]), deconvolution of bulk spatial transcriptomics profiles (Stereoscope[15] or DestVI[16]), doublet detection (Solo[17]) and multi-modal analysis of CITE-seq (cellular indexing of transcriptomes and epitopes by sequencing) data (totalVI[18]).

In the broader analysis pipeline, scvi-tools sits downstream of initial quality control (QC)-driven preprocessing and generates outputs that may be further interpreted via general single-cell analysis tools (Fig. 1a). At its core, scvi-tools implements several key functionalities that are accessible across data modalities, such as differential analysis and dataset integration. All 14 models (Supplementary Table 1) currently implemented in scvi-tools interact with Scanpy through the annotated dataset (AnnData[19]) format, and the models share a consistent user interface (Fig. 1b). The scvi-tools library also has an interface with R such that each model may be used in Seurat or Bioconductor pipelines.

We also illustrate two new features of scvi-tools applicable to several types of omics data. The first feature offers the ability to remove unwanted variation due to multiple nuisance factors simultaneously, including both discrete (for example, batch category) and continuous (for example, percent mitochondrial reads) factors. In Supplementary Note 2, we apply this in the context of an scRNA-seq dataset of *Drosophila* wing development that suffered from nuisance variation due to cell cycle, sex and replicate. The second feature extends several scvi-tools integration methods to iteratively integrate new 'query' data into a pretrained 'reference' model via the recently proposed scArches neural network architecture surgery[12]. This feature is particularly useful for incorporating new samples into an analysis without having to reprocess the entire set of samples. Supplementary Note 3 presents a case study of applying this approach with totalVI by projecting data from patients with COVID-19 into an atlas of immune cells.

From the perspective of a methods developer, scvi-tools offers a set of building blocks that make it easy to implement new models and modify existing models with minimal code overhead (Fig. 2a,b and Supplementary Note 4). These building blocks use popular libraries, such as AnnData[12], PyTorch[20], PyTorch Lightning[9] and Pyro[21], and facilitate probabilistic model design with neural network components and GPU acceleration. This allows method developers to primarily focus on developing probabilistic models instead of on data management, model training and user-interface code. We demonstrate how these building blocks can be used for efficient model development through a reimplementation of Stereoscope, in which we demonstrate a substantial reduction in code complexity (Fig. 2c–e and Supplementary Note 5). This example demonstrates the broad scope of analyses that may be powered by scvi-tools.

On the scvi-tools documentation website, we feature the application programming interface (API) reference of each model, as well as tutorials describing the functionality of each model and its interaction with other single-cell tools. We also make these tutorials available via Google Colab, which provides a free computing environment and GPU and can even support large-scale analyses.

In the development of scvi-tools, we aimed to bridge the gap that exists between the single-cell software ecosystem and the contemporary machine-learning frameworks for constructing and deploying this class of models. Thus, developers can now expect to build models that are immediately accessible to end users in the single-cell community while continuing to rely on popular machine-learning libraries. On our documentation website, we provide a series of tutorials on building a model with scvi-tools, walking through the steps of data management, module construction and model development. We also built a template repository on GitHub that enables developers to quickly create a Python package that uses unit testing, automated documentation and popular code styling libraries. This repository demonstrates how the scvi-tools building blocks can be used for external model deployment. We anticipate that most models built with scvi-tools will be deployed in this way as independent packages while adhering to standard API and coding conventions, which will make them more readily accessible for new users.

As scvi-tools remains under active development, end users can expect that scvi-tools will continually evolve, adding support for new models, new workflows and new features. We anticipate that these resources will serve the single-cell community by facilitating the prototyping of new models, creating a standard for the deployment of probabilistic analysis software and enhancing the scientific discovery pipeline. ❐

Adam Gayoso[1,23], Romain Lopez[2,23], Galen Xing[1,3,23], Pierre Boyeau[2,4], Valeh Valiollah Pour Amiri[1,2], Justin Hong[1,2], Katherine Wu[2], Michael Jayasuriya[2], Edouard Mehlman[4,5], Maxime Langevin[5,18,19], Yining Liu[2,20], Jules Samaran[6], Gabriel Misrachi[5,21], Achille Nazaret[5,20], Oscar Clivio[4,22], Chenling Xu[1], Tal Ashuach[1], Mariano Gabitto[1,2], Mohammad Lotfollahi[7,8], Valentine Svensson[9], Eduardo da Veiga Beltrame[10], Vitalii Kleshchevnikov[11], Carlos Talavera-López[11,12], Lior Pachter[10,13], Fabian J. Theis[7,8], Aaron Streets[1,3,14], Michael I. Jordan[1,2,15], Jeffrey Regier[16] and Nir Yosef[1,2,3,17] ✉

[1]Center for Computational Biology, University of California, Berkeley, Berkeley, CA, USA. [2]Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, Berkeley, CA, USA. [3]Chan Zuckerberg Biohub, San Francisco, CA, USA. [4]École Normale Supérieure Paris-Saclay, Gif-sur-Yvette, France. [5]Centre de Mathématiques Appliquées, École polytechnique, Palaiseau, France. [6]Mines Paristech, PSL University, Paris, France. [7]Institute of Computational Biology, Helmholtz Center Munich, Munich, Germany. [8]School of Life Sciences Weihenstephan, Technical University of Munich, Munich, Germany. [9]Serqet Therapeutics, Cambridge, MA, USA. [10]Division of Biology and Biological Engineering, California Institute of Technology, Pasadena, CA, USA. [11]Cellular Genetics Programme, Wellcome Sanger Institute, Wellcome Genome Campus, Cambridge, UK. [12]EMBL-European Bioinformatics Institute, Wellcome Genome Campus, Cambridge, UK. [13]Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA, USA. [14]Department of Bioengineering, University of California, Berkeley, Berkeley, CA, USA. [15]Department of Statistics, University of California, Berkeley, Berkeley, CA, USA. [16]Department of Statistics, University of Michigan, Ann Arbor, MI, USA. [17]Ragon Institute of Massachusetts General Hospital, MIT and Harvard, Cambridge, MA, USA. [18]Present address: Pasteur, Department of Chemistry, École Normale Supérieure, PSL University, Paris, France. [19]Present address: Molecular Design Sciences – Integrated Drug Discovery, Sanofi R&D, Vitry-sur-Seine, France. [20]Present address: Department of Computer Science, Columbia University, New York, NY, USA. [21]Present address: Gleamer, Paris, France. [22]Present address: Department of Statistics, University of Oxford, Oxford, UK. [23]These authors contributed equally: Adam Gayoso, Romain Lopez and Galen Xing

✉*e-mail:* *niryosef@berkeley.edu*

Published online: 7 February 2022
https://doi.org/10.1038/s41587-021-01206-w

References

1. Svensson, V., da Veiga Beltrame, E. & Pachter, L. *Database* **2020**, baaa073 (2020).
2. Lee, J., Hyeon, D. Y. & Hwang, D. *Exp. Mol. Med.* **52**, 1428–1442 (2020).
3. Wagner, A., Regev, A. & Yosef, N. *Nat. Biotechnol.* **34**, 1145–1160 (2016).
4. Zappia, L., Phipson, B. & Oshlack, A. *PLOS Comput. Biol.* **14** (2018).
5. Lopez, R., Gayoso, A. & Yosef, N. *Mol. Syst. Biol.* **16**, e9198 (2020).
6. Gentleman, R. C. et al. *Genome Biol.* **5**, R80 (2004).
7. Satija, R., Farrell, J. A., Gennert, D., Schier, A. F. & Regev, A. *Nat. Biotechnol.* **33**, 495–502 (2015).
8. Wolf, F. A., Angerer, P. & Theis, F. J. *Genome Biol.* **19**, 15 (2018).
9. Falcon, W. & The PyTorch Lightning team. PyTorch Lightning (Version 1.4). (2019); https://doi.org/10.5281/zenodo.3828935
10. Chollet, F. et al. Keras. https://keras.io (2015).
11. Lopez, R., Regier, J., Cole, M. B., Jordan, M. I. & Yosef, N. *Nat. Methods* **15**, 1053–1058 (2018).
12. Lotfollahi, M. et al. *Nat. Biotechnol.* **40**, 121–130 (2022).
13. Zhang, A. W. et al. *Nat. Methods* **16**, 1007–1015 (2019).
14. Xu, C. et al. *Mol. Syst. Biol.* **17**, e9620 (2021).
15. Andersson, A. et al. *Commun. Biol.* **3**, 565 (2020).
16. Lopez, R. et al. Preprint at bioRxiv https://doi.org/10.1101/2021.05.10.443517 (2021).
17. Bernstein, N. J. et al. *Cell Syst.* **11**, 95–101.e5 (2020).
18. Gayoso, A. et al. *Nat. Methods* **18**, 272–282 (2021).
19. Angerer, P., Wolf, A., Virshup, I. & Rybakov, S. AnnData. *GitHub* https://github.com/theislab/anndata (2019).
20. Paszke, A. et al. *Adv. Neural Inf. Process. Syst.* **32**, 8026–8037 (2019).
21. Bingham, E. et al. *J. Mach. Learn. Res.* **20**, 1–6 (2019).

Acknowledgements

We acknowledge members of the Streets and Yosef laboratories for general feedback. We thank all the GitHub users who contributed code to scvi-tools over the years. We thank Nicholas Everetts for help with the analysis of the *Drosophila* data. We thank David Kelley and Nick Bernstein for help implementing Solo. We thank Marco Wagenstetter and Sergei Rybakov for help with the transition of the scGen package to use scvi-tools, as well as feedback on the scArches implementation. We thank Hector Roux de Bézieux for insightful discussions about the R ecosystem. We thank Kieran Campbell and Allen Zhang for clarifying aspects of the original CellAssign implementation. We thank the Pyro team, including Eli Bingham, Martin Jankowiak and Fritz Obermeyer, for help integrating Pyro in scvi-tools. Research reported in this manuscript was supported by the NIGMS of the National Institutes of Health under award number R35GM124916 and by the Chan-Zuckerberg Foundation Network under grant number 2019-02452. O.C. is supported by the EPSRC Centre for Doctoral Training in Modern Statistics and Statistical Machine Learning (EP/S023151/1, studentship 2420649). A.G. is supported by NIH Training Grant 5T32HG000047-19. A.S. and N.Y. are Chan Zuckerberg Biohub investigators.

Author contributions

A.G., R.L and G.X. contributed equally. A.G. designed the scvi-tools application programming interface with input from G.X. and R.L. G.X. and A.G. led development of scvi-tools with input from R.L. G.X. reimplemented scVI, totalVI, AutoZI and scANVI with input from A.G. R.L. implemented Stereoscope with input from A.G. Data analysis in this manuscript was led by A.G., R.L. and G.X, with input from N.Y. A.G., R.L., P.B., E.M., M. Langevin., Y.L., J.S., G.M. and A.N., O.C. worked on the initial version of the codebase (scvi package), with input from M.I.J, J.R. and N.Y. R.L., E.M. and C.X. contributed the scANVI model, with input from J.R. and N.Y. A.G. implemented totalVI with input from A.S. and N.Y. T.A. implemented peakVI with input from A.G. A.G implemented scArches with input from M. Lotfollahi., F.J.T and N.Y. V.S. made several contributions to the codebase, including the LDVAE model. P.B. contributed the differential expression programming interface. E.d.V.B. and C.T.-L. provided tutorials on differential expression and deconvolution of spatial transcriptomics, with input from L.P. K.W. implemented CellAssign in the codebase with input from A.G. V.V.P.A., J.H. and M.J. made general code contributions and helped maintain scvi-tools. J.H. implemented LDA. T.A. and M.G. implemented MultiVI. V.K. improved Pyro support in scvi-tools and ported Cell2Location to use scvi-tools. N.Y. supervised all research. A.G., R.L., G.X., J.R. and N.Y. wrote the manuscript.

Competing interests

V.S. is a full-time employee of Serqet Therapeutics and has ownership interest in Serqet Therapeutics. F.J.T. reports consulting fees from Roche Diagnostics GmbH and Cellarity Inc., and ownership interest in Cellarity, Inc. N.Y. is an advisor to and/or has equity in Cellarity, Celsius Therapeutics and Rheos Medicines. The remaining authors declare no competing interests.

Additional information

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s41587-021-01206-w.