nature portfolio

https://doi.org/10.1038/s41587-021-01206-w

Supplementary information

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

In the format provided by the authors and unedited

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

SUPPLEMENTARY INFORMATION

Adam Gayoso [1,*], Romain Lopez [2,*], Galen Xing [1,3,*], 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,A,B,†], Yining Liu [2,C], Jules Samaran [6,†], Gabriel Misrachi [5,D,†], Achille Nazaret [5,C,†], Oscar Clivio [4,E,†], 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, USA
[2] Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, Berkeley, USA
[3] Chan Zuckerberg Biohub, San Francisco, 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, USA
[10] Division of Biology and Biological Engineering, California Institute of Technology, Pasadena, USA
[11] Cellular Genetics Programme, Wellcome Sanger Institute, Wellcome Genome Campus, Hinxton, Cambridge, UK
[12] EMBL - EBI, Wellcome Genome Campus, Hinxton, Cambridge, UK
[13] Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena, USA
[14] Department of Bioengineering, University of California, Berkeley, Berkeley, USA
[15] Department of Statistics, University of California, Berkeley, Berkeley, USA
[16] Department of Statistics, University of Michigan, Ann Arbor, USA
[17] Ragon Institute of MGH, MIT and Harvard, Cambridge, USA

Present addresses:
[A] Pasteur, Department of Chemistry, École Normale Supérieure, PSL University, Paris, France
[B] Molecular Design Sciences - Integrated Drug Discovery, Sanofi R&D, Vitry-sur-Seine, France
[C] Department of Computer Science, Columbia University, New York, USA
[D] Gleamer, Paris, France
[E] Department of Statistics, University of Oxford, Oxford, UK

* These authors contributed equally.
† Work done while interning in the Yosef Lab, UC Berkeley
# Corresponding author: niryosef@berkeley.edu

Contents

Supplementary Note 1

scvi-tools is a unified resource for single-cell omics data analysis

The single-cell omics data analysis pipeline is composed of several steps [1, 2] (Figure 1a). First, data are staged within a data object using packages like Scanpy [3], Seurat [4] or scater [5] and preprocessed with quality control filters. The data are then analyzed through a variety of subsequent steps, which aim to normalize, simplify, infer structure, annotate, and extract new insight (Figure 1b). scvi-tools aims to provide a rich set of methods for these latter steps, while relying on probabilistic models for statistically sound interpretation of data.

The models currently implemented in scvi-tools can perform normalization, dimensionality reduction, dataset integration, differential expression (scVI [6, 7], scANVI [8], totalVI [9], PeakVI [10], LDVAE [11], MultiVI [12]), automated annotation (scANVI, CellAssign [13]), doublet detection (Solo [14]), factor analysis (LDVAE, LDA [15]), and deconvolution of spatial transcriptomics profiles (Stereoscope [16], DestVI [17]). These models span multiple modalities including scRNA-seq (scVI, scANVI, CellAssign, Solo), CITE-seq [18] (totalVI), single-cell ATAC-seq (PeakVI), spatial transcriptomics (Stereoscope, gimVI [19], DestVI [17]), and combinations thereof (multiome; MultiVI) (Supplementary Table 1). Importantly, these models make use of stochastic inference techniques and (optionally) GPU acceleration, such that they readily scale to even the largest datasets.

Each model also comes with a simple and consistent application programming interface (API). They all rely on the popular AnnData format as a way to store and represent the raw data (Figure 1a). Consequently, scvi-tools models are easily integrated with Scanpy workflows. This also enables users to interface with AnnData-based data zoos like Sfaira [20]. Furthermore, the globally consistent API of scvi-tools allows us to maintain a reticulate-based [21] workflow in R, such that scvi-tools models may be used directly in Seurat or Bioconductor workflows. Therefore, after running methods in scvi-tools, results can be visualized and further assessed with a broad range of analysis packages like Scanpy, Seurat, VISION [22], and cellxgene [23].

Supplementary Note 2

Nonlinear removal of unwanted variation due to multiple covariates

As the quantity, size, and complexity of single-cell datasets continues to grow, there is a significant need for methods capable of controlling for the effects of unwanted variation [24]. Some factors that contribute to unwanted variation depend directly on the data generating process, such as differences between labs, protocols, technologies, donors, or tissue sites. Normally, such nuisance factors are observable and available as sample-level metadata. Unwanted variation can also come at the level of a single cell and can be calculated directly from the data. It can stem from technical factors like quality as gauged by proxies such as the abundance of mitochondrial RNA or the expression of housekeeping genes, as well as biological factors like cell cycle phase. Nuisance factors can come in either categorical or numerical form. These factors can affect the data in a nonlinear manner [25], and controlling for them is essential for most forms of downstream analysis.

Many methods have been proposed for removing unwanted variation, but most target the subtask of dataset integration, which consists of controlling for one categorical confounding factor at the sample-level, like sample ID [27, 28]. Harmony [29] is, to the best of our knowledge, the only method capable of nonlinearly controlling for multiple categorical covariates simultaneously. While the focus of the integration task is on categorical factors, some pipelines provide an additional layer of normalization, where a given numerical confounder (typically, though not limited to the cell-level) can be regressed out prior to batch correction using a linear model (e.g, the "regress out" function in Scanpy [3] or Combat [30]). Thus, no existing method is capable of performing nonlinear removal of unwanted variation with respect to multiple (cell- or sample-level) categorical and continuous covariates. However, we anticipate an increase in the need for such methods, reflecting the complexity of recent single-cell atlases.

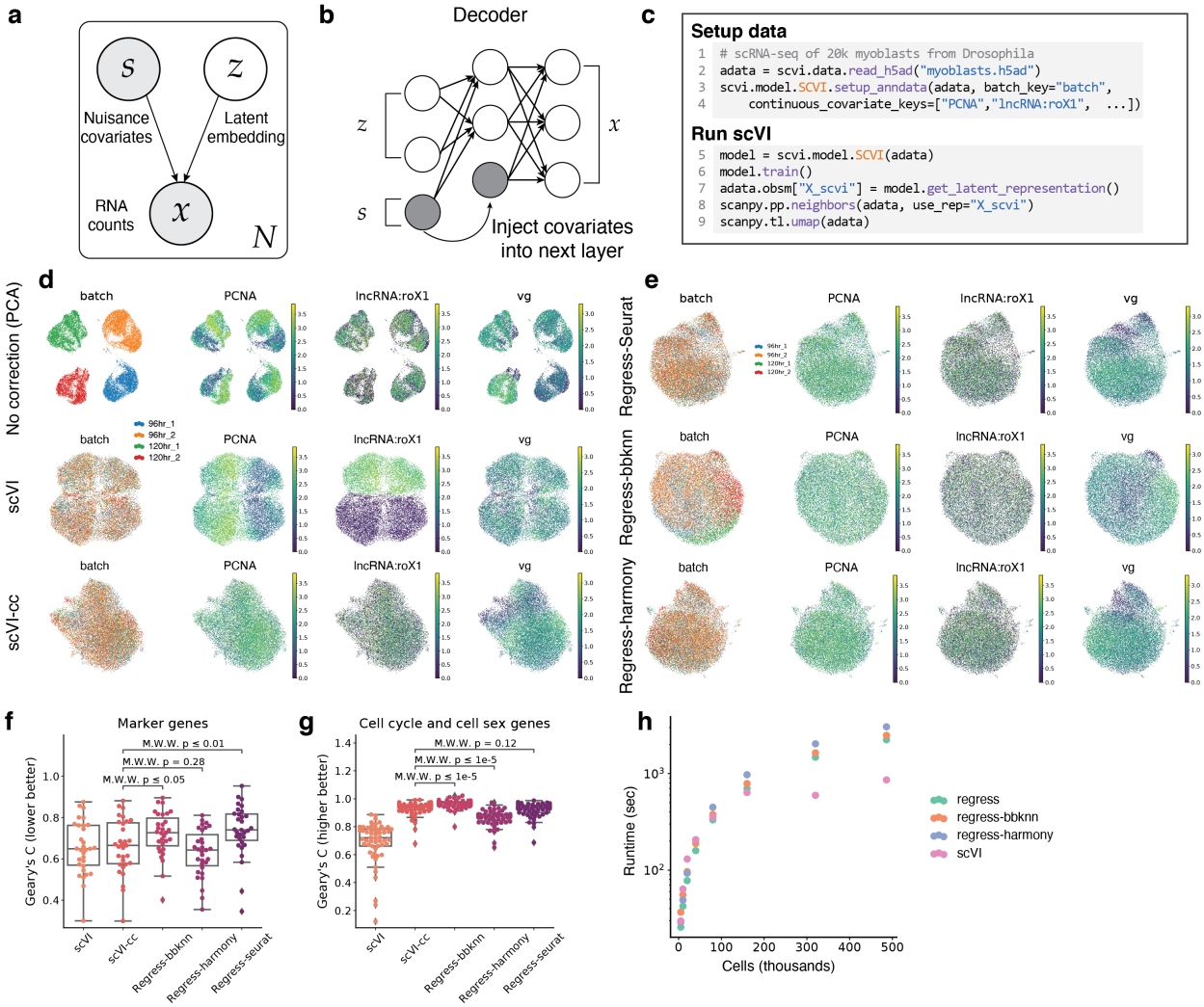
Our previously described models scVI [6], scANVI [8], totalVI [9], and PeakVI [10] all rely on a latent variable models and neural networks to remove unwanted variation from observed covariates, denoted as s_n for a cell n (Supplementary Figure 1a). These models learn a latent representation of each cell that is corrected for this unwanted variation by using a nonlinear neural network decoder that receives the representation and the observed covariates as input, and subsequently injects the covariates into each hidden layer (Supplementary Figure 1b). While the models could theoretically process multiple categorical or continuous covariates, their previous implementations restricted this capability due to the difficulty of implementing proper data management throughout the training and downstream analysis of a fitted model. scvi-tools allowed us to address this obstacle and support conditioning on arbitrary covariates via its global data registration process (`setup_anndata`; Supplementary Note 4) and shared neural network building blocks. Now, users simply register their covariates before running a model (Supplementary Figure 1c). By implementing this feature in the four aforementioned models, scvi-tools can handle complex data collections across a range of modalities and analysis tasks. Additionally, this feature can be easily propagated to new models (and modalities), all with the same user experience, due to the modular nature of the scvi-tools package.

To demonstrate this capability, we applied scVI to a dataset of Drosophila wing disc myoblast cells [31] that suffered from strong effects due to cell cycle and sex of the donor organism, both of which were observed via a set of nuisance genes (i.e., numerical covariates; Supplementary Table 2). Both of these confounding factors are observed at the cell-level, as these cells were taken from batches of fly larva and were processed together without sex sorting. The dataset also featured a sample-level confounding factor (batch ID; two batches) and non-nuisance factor (developmental time point; Supplementary Figure 1d).

When scVI was trained only conditioned on the batch covariate, we observed that it successfully integrated each batch, however the effects from *PCNA*, a cell cycle gene, and *IncRNA:roX1*, a gene that is expressed by males (Supplementary Figure 1d), still manifested in the latent space. When scVI was additionally conditioned on the expression of the set of nuisance genes (scVI-cc; Methods), we observed that it successfully integrated the data across batch, cell cycle, and sex. Additionally, we found that the desirable biological signals, such as the expression of *vg*, a gene marking a spatial compartment in the wing disc, were preserved (Supplementary Figure 1d).

We compared these results to the respective Seurat-based and Scanpy-based workflow, which consisted of the `scanpy.pp.regress_out` function for linear removal of signal from nuisance genes followed

by Seurat [32] (regress-seurat), bbknn [33] (regress-bbknn) or Harmony (regress-harmony) for correction of batch effects (Supplementary Figure 1e; Methods). We evaluated these alternative workflows using an autocorrelation measure (Geary's C [34]; Methods) computed with respect to each workflow's low-dimensional representation and a set of genes. This metric quantifies the extent to which the gene expression of a cell can be explained by its neighboring cells. An optimal correction would leave a set of key marker genes well explained by the neighborhood graph (low Geary's C) while forcing nuisance genes to not be well explained by the same graph structure (high Geary's C). We observed that scVI-cc was able to both retain important biological signal (via the set of key marker genes) and remove the unwanted variation due to the nuisance genes relative to the vanilla scVI baseline (Supplementary Figure 1f,g). While we found a tradeoff between retaining biological signal of marker genes and removing nuisance variation across all workflows, these results demonstrate that scVI strikes a balance in removing unwanted variation and retaining wanted variation. The scVI workflow with multiple covariates is also more scalable than the alternatives; we evaluated scalability using subsampled versions of the Heart Cell Atlas dataset [35] that had categorical covariates like donor and continuous covariates like cell-level mitochondrial count percentage (Supplementary Figure 1h; Methods).



Supplementary Figure 1: Removal of unwanted variation in the analysis of Drosophila wing disc development. **a**, Graphical model representation of a latent variable models in scvi-tools that conditions on nuisance covariates. **b**, Graphical representation of covariates injected into each layer of decoder neural networks. **c**, Code snippet to register AnnData and train scVI with continuous covariates. The covariates are identified with keys stored in the `AnnData.obs` cell-level data frame. **d, e**, UMAP [26] embedding of **(d)** uncorrected latent space computed using principal components analysis, scVI latent space with only batch covariates (scVI), scVI latent space with batch and continuous covariates (scVI-cc), **(e)** low-dimensional embeddings after running `scanpy.pp.regress_out` followed by Seurat (Regress-seurat), bbknn (Regress-bbknn), and similarly after running `scanpy.pp.regress_out` followed by Harmony (Regress-harmony). UMAP plots are colored by batch, *PCNA* (cell cycle gene), *lncRNA:roX1* (cell sex gene), and *vg* (gene marking spatial compartment within the wing disc). **f**, Geary's C of canonical marker genes of interest per model. Hypothesis testing between scVI-cc and competing mehods was performed with a Mann Whitney Wilcoxon test (two-sided; M.W.W). **g**, Geary's C of the cell cycle and cell sex genes conditioned on per model. Box plots were computed on n=31 genes for **(f)** and n=55 genes for **(g)** and indicate the median (center lines), interquartile range (hinges), and whiskers at 1.5× interquartile range. Gene lists can be found in Supplementary Table 2. **h**, Runtime of workflows to correct multiple continuous and categorical covariates on subsampled versions of the Heart Cell Atlas dataset.

Methods

Drosophila The dataset of Drosophila myoblasts was downloaded from Gene Expression Omnibus (Accession ID GSE155543). The list of cell cycle and cell sex genes were derived from ref. [31] while marker genes were derived from refs. [31] and [36]. (Supplementary Table 2). These marker genes were previously identified as markers of either the direct or indirect myoblasts, subpopulations of the wing disc-associated myoblasts in the larva that eventually give rise to distinct adult flight muscles. scVI was trained on the raw count data with a single batch covariate for timepoint and replicate. For scVI-cc, the model was conditioned on each nuisance gene's log normalized expression (first each cells counts were normalized, then scaled to the median of total counts for cells before normalization, before calculating the log on the scaled normalized counts) as well as a categorical batch covariate for timepoint and replicate. Furthermore, each nuisance gene that was conditioned on was also removed from the input count matrix. Each scVI model was trained for 400 epochs. For regress-bbknn, the `regress_out` scanpy function was used for the continuous covariates, while the `bbknn` scanpy function was used for the categorical batch covariates. Similar to scVI-cc, the nuisance gene expression was removed from the input count matrix. For regress-harmony, the `regress_out` function was used for the continuous covariates, while the `harmony_integrate` scanpy function was used for the categorical batch covariates. The nuisance genes were also removed from the input count matrix. For regress-seurat, the `regress_out` function was used to correct for continuous covariates. Then `anndata2ri` (version 1.0.6, https://github.com/theislab/anndata2ri) was used to convert the AnnData to a SingleCellExperiment object. Then to correct for the categorical covariates, we used (with Seurat version 4.0.1) `SelectIntegrationFeatures` to select features variable across our categorical covariates, followed by `FindIntegrationAnchors`, and `IntegrateData` in order to create an integrated dataset across all categorical covariates. Finally `RunPCA` was used to get the final latent representation. To compute Geary's C we used the `gearys_c` scanpy function.

Runtime Runtime analysis was run on a desktop with an Intel Core i9-10900K 3.7 GHz processor, 2x Corsair Vengeance LPX 64GB ram, and an NVIDIA RTX 3090 GPU. Runtime was performed with the Heart Cell Atlas dataset downloaded from https://www.heartcellatlas.org/. We then subsampled and created datasets of 5,000, 10,000, 20,000, 40,000, 80,000, 160,000, 320,000, and 486,134 cells and selected the top 4,000 genes via `highly_variable_genes` per dataset, with parameter `flavor="seurat_v3"`. For each of the following methods, we used the cell_source and `donor` fields of the dataset as categorical covariates. For continuous covariates, we generated 8 random covariates by sampling from a standard normal distribution in addition to treating the percent_mito and percent_ribo fields as additional continuous covariates, for a total of 10 continuous covariates. For scVI runtime, we tracked the runtime of running the `train` function with the following parameters:

```
early_stopping=True, early_stopping_patience=45, max_epochs=10000,
batch_size=1024, limit_train_batches=20,
train_size=0.9 if n_cells < 200000 and train_size=1-(20000/n_cells) otherwise
```

This choice of the size parameter for the training set ensures that the validation set always has size of less than 20,000 cells for the whole runtime experiment. For the regress-bbknn baseline, we tracked the runtime of correcting continuous covariates with `regress_out`, running `pca`, and correcting for categorical covariates with `bbknn` (all from the scanpy package). For the regress-harmony baseline, we tracked the runtime of `regress_out` to correct for continuous covariates, running `pca`, and correcting for categorical covariates with `harmony_integrate` (all from the scanpy package). For the regress baseline, we only tracked the runtime of the `regress_out` function.

Supplementary Note 3

Reference mapping with scArches

While dataset integration provides a way to leverage information from many sources, current methods do not scale well to the subtask of integration in which a "query" dataset is integrated with a large, annotated "reference" dataset. This is an increasingly common scenario, however, that is driven by community efforts for establishing consolidated tissue atlases. These atlases are meant to be used as general references of cell states in a given tissue and may consist of millions of cells [37].

scArches [38] is a recent method that was developed to address this scenario. scArches leverages conditional (variational) autoencoders and transfer learning to decouple the reference mapping task into two subproblems: First, a reference model is trained on the reference data only; and second, the neural network from the reference model is augmented with nodes that are only influenced by the query data. This new part of the network is subsequently trained with the query data, resulting in a joint model that describes both the datasets, while correcting for the technical variation between the query and reference (reference is unchanged). This procedure dramatically reduces the computational burden of dataset integration.

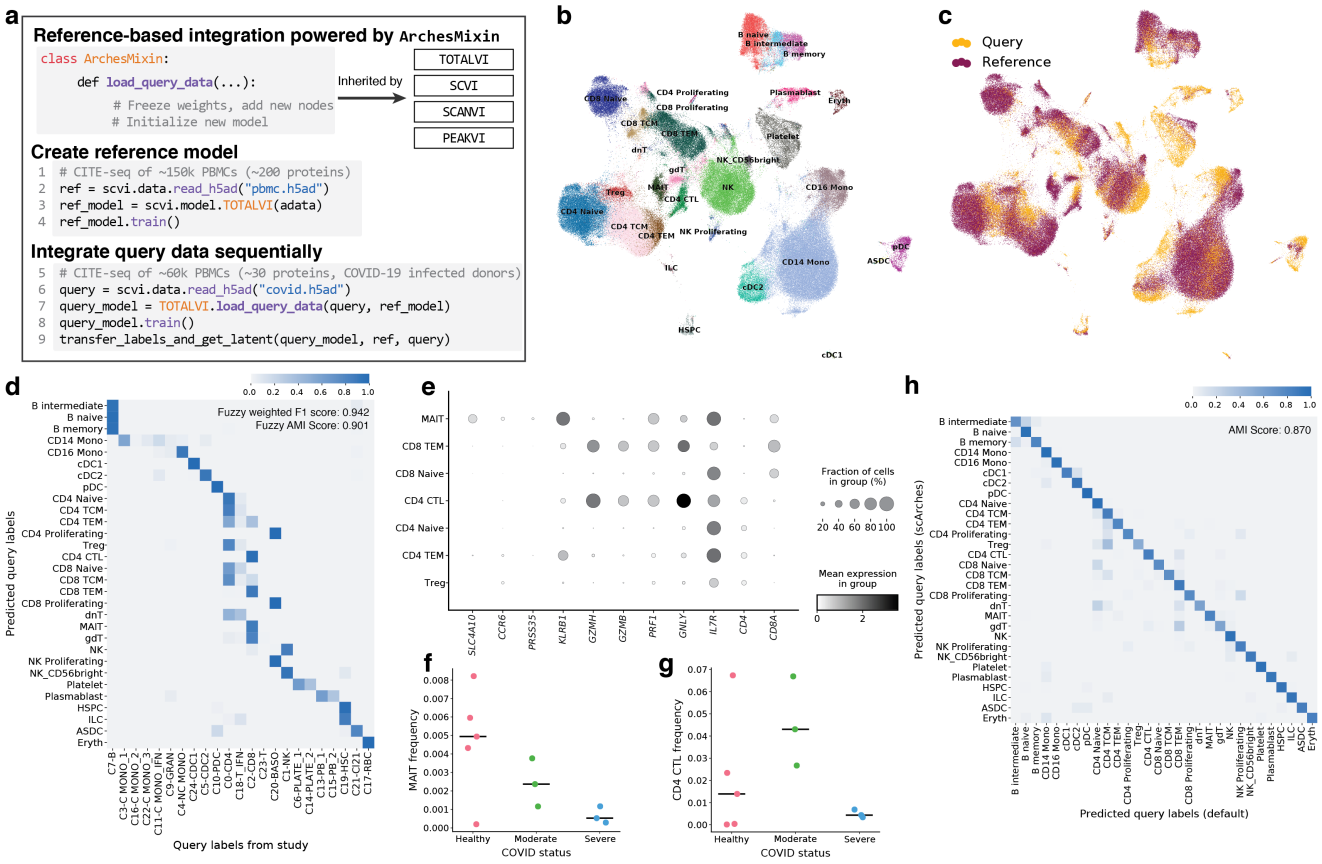
Assuming a pre-trained reference model is available (e.g., representing the "atlas" of cell state for a particular tissue), one only needs to process the (typically much smaller) query data, which makes scArches a purely online procedure. This is in contrast to other reference mapping methods like Seurat [39, 32], which requires the presence of the reference data to correct technical variation and transfer metadata. Furthermore, scArches-style reference mapping can better leverage model/data sharing repositories like Sfaira [20], as the model file itself is approximately 15 megabytes. Beyond the aforementioned reference mapping scenario, scArches is also useful in studies where data is accumulated gradually as it eliminates the need for reanalysis when new samples are collected.

We implemented the scArches method in scvi-tools through the addition of one class called `ArchesMixin`. This class contains a generic procedure for adding new nodes to a given reference model, as well as appropriately freezing the nodes corresponding to the reference dataset during training. The `ArchesMixin` class is already inherited by scVI, scANVI, totalVI, and peakVI, without any other custom code, and it can easily be inherited by new models. From a user's perspective, this inheritance adds one function called `load_query_data` to each of these models that is used to load a pre-trained reference model with new query data.

We applied scArches-style reference mapping in the multi-modal CITE-seq setting with totalVI in order to quickly annotate and then interpret a dataset of immune cells of the blood in donors responding to COVID-19 infection [40] (Supplementary Figure 2a, Methods). First, we used totalVI to train a reference model of immune cell states, using an annotated CITE-seq dataset of 152,094 peripheral blood mononuclear cells (PBMCs) with over 200 surface proteins [32]. Next, we applied scArches to augment the reference model with an additional CITE-seq dataset of 57,669 PBMCs with 35 proteins from donors with moderate and severe COVID-19, as well as healthy controls [40]. We note that scArches-powered totalVI is to the best of our knowledge the only method that can perform reference mapping in the case where both reference and query are CITE-seq datasets. The most similar method, Seurat v4 [32], can use CITE-seq protein data to build the reference, but can only use scRNA-seq data for the query.

Running totalVI in this setting is made straightforward with the scvi-tools interface and required only a few lines of code (Supplementary Figure 2a). After query training, we visualized the joint latent representation of the reference and query cells using UMAP (Supplementary Figure 2b,c). We transferred the annotated cell type labels from the reference cells to the query cells using a random forest classifier operating on the 20-dimensional joint latent space. As the model does not change for the reference data, the training of the classifier is independent of the query data. Therefore, the classifier itself can also be seen as a part of the pre-trained reference model.

The predicted labels of the query cells generally agreed with the labels that were provided in the original study and held-out from this analysis (Supplementary Figure 2d). However, there were some inconsistencies related to T cell subtype classification. The totalVI-scArches approach identified populations of Mucosal associated invariant T (MAIT) cells and CD4 positive cytotoxic T lymphocytes (CD4 CTLs), whereas both populations were mostly annotated as CD8 T cells in the original study

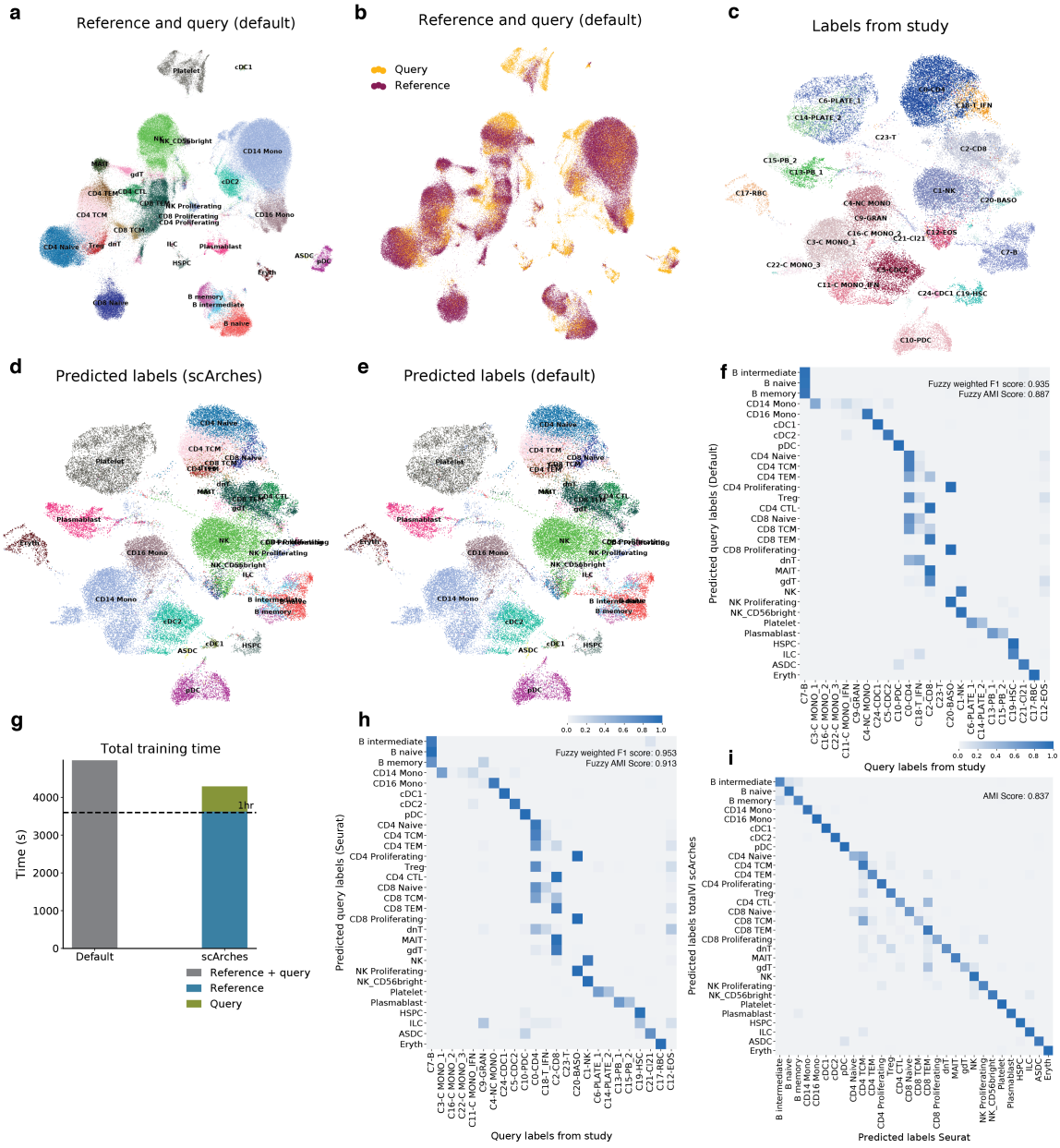


Supplementary Figure 2: Sequential integration of CITE-seq PBMC samples with totalVI and the scArches method. a, (Pseudo)code-based overview of using scArches with the implementation of totalVI in scvi-tools. scArches was implemented globally through the `ArchesMixin` class. First, the reference model is trained on reference data, and then the scArches architectural surgery is performed when `load_query_data` is called on the query data. Finally, the (now) query model is trained with the query data and downstream analysis is performed. **b, c**, UMAP embedding of the totalVI reference and query latent spaces colored by **(b)** the reference labels and predicted query labels and **(c)** the dataset of origin. **d**, Row-normalized confusion matrix of scArches predicted query labels (rows) and study-derived cell annotations (columns). **e**, Dotplot of log library size normalized RNA expression across cell type markers for predicted T cell subsets. **f, g** Frequency of **(f)** MAIT cells and **(g)** CD4 CTLs for each donor in the query dataset across healthy controls and donors with moderate and severe COVID. Horizontal line denotes median. **h**, Row-normalized confusion matrix of scArches predicted query labels (rows) and default totalVI predicted labels (columns).

(Supplementary Figure 2d). We found that the predicted MAIT subpopulation had expression of the known markers *SLC4A10*, *CCR6*, *KLRB1*, and *PRSS35*, and that they decreased in frequency as a function of COVID severity, which has been previously described [41, 42] (Supplementary Figure 2e, f). CD4 CTLs have not been as well characterized in terms of their response to COVID, but the predicted CD4 CTLs had relatively high expression of cytotoxic molecules like *PRF1*, *GZMB*, *GZMH*, *GNLY*, and were found to be most prevalent in donors with moderate COVID (Supplementary Figure 2e,g). This pattern is consistent with evidence suggesting that the presence of CD4 CTLs is associated with better clinical outcomes in other viral infections in humans [43], though more targeted study designs may be necessary to better understand this relationship [44]. Overall, these results suggest that integrating with reference atlases can lead to more rapid, and potentially more accurate and consistent annotations of cells across studies.

Finally, we compared the totalVI-scArches approach to *default* totalVI, namely training totalVI in one step, using both the reference and query datasets, as well as to the Seurat v4 workflow. Using the same random forest procedure, we found that the predictions from totalVI scArches and default modes were in high agreement (Supplementary Figure 2h, Supplementary Figure 3a-e), with the scArches mode

producing slightly more accurate predictions (Supplementary Figure 3f; Methods). This is despite the massive speed increase of the scArches approach, which took only 10 minutes to integrate the query dataset, whereas default totalVI took over 80 minutes total due to necessary retraining with the reference data (Supplementary Figure 3g). Comparing totalVI scArches to Seurat v4, we observed that the methods had similar performance with Seurat v4 having the slight advantage (Supplementary Figure 2d, Supplementary Figure 3h,i). We note that the cell type labels in the reference dataset were derived from Seurat v4, which should provide Seurat an inherent advantage over totalVI in this particular example.



Supplementary Figure 3: Sequential integration of PBMC samples with totalVI and the scArches method. **a, b**, UMAP embedding of the totalVI reference and query latent spaces from default totalVI model run colored by **(a)** the reference labels and predicted query labels and **(b)** the dataset of origin. **c-e**, UMAP embedding of the totalVI scArches query latent space from colored by **(c)** study-derived labels, **(d)** predicted labels from totalVI scArches, and **(e)** predicted labels from totalVI default. **f**, Row-normalized confusion matrix of totalVI default predicted query labels (rows) and study-derived cell annotations (columns), **g**, Runtime of totalVI default versus totalVI scArches, **h**, Row-normalized confusion matrix of Seurat v4 predicted query labels (rows) and study-derived cell annotations (columns), **i**, Row-normalized confusion matrix of totalVI scArches predicted labels (rows) and Seurat predicted labels (columns).

Methods

The reference dataset of Human PBMCs corresponding to the CITE-seq dataset described in [32] was downloaded from Gene Expression Omnibus (Accession ID GSE164378). Associated metadata, like cell-type annotations, were retrieved from https://atlas.fredhutch.org/nygc/multimodal-pbmc/. The query dataset of Human PBMCs corresponding to the CITE-seq dataset described in ref. [40] was provided by the authors, with full metadata including study-derived annotations and other study design information. The reference data were additionally filtered to include cells that meet the following criteria: (1) greater than 150 proteins detected, (2) percent mitochondrial counts less than 12%, (3) not doublets (i.e., reference cells annotated as doublets removed), (4) natural log protein library size, defined as total protein counts, between 7.6 and 10.3. Furthermore, protein features corresponding to isotype controls were removed (for reference and query), and protein features targeting the same protein (i.e., antibody clones) were summed together (applies to reference only). The query data were additionally filtered for doublets using Scrublet [45] with default parameters. Highly variable genes (4000) were selected using only the reference datasets and with the method in Scanpy (`flavor="seurat_v3"`). Finally, the protein expression for a random set of five out of the 24 batch categories (representing time and donor) in the reference dataset were masked from totalVI, in order to help the model generalize to query data with mismatched proteins.

In the case of scArches, totalVI was trained on the reference data for 250 epochs, using two hidden layers and layer normalization as described in the totalVI scArches tutorial https://docs.scvi-tools.org/en/stable/user_guide/notebooks/scarches_scvi_tools.html. After applying the scArches architectural surgery, the model was updated with the query data for 150 additional epochs. The latent representation for the reference and query datasets was obtained from the model after it was updated with the query data; however, we note that the latent representation of the reference data does not change after query traning in the case of scArches. In the case of default totalVI, we kept all hyperparameters and data preprocessing the same, except that we trained totalVI only once, on the concatenated reference and query datasets for 250 epochs.

For both scArches and default totalVI, we transferred the reference cell-type annotations using a random forest classifier implemented in scikit-learn [46]. In particular, we trained a random forest classifier on the latent representations of the reference cells (default parameters except `class_weight="balanced_subsample"`). The query cell annotations were then obtained by passing the query latent representations to the classifier. In all cases, UMAP embeddings were computed using Scanpy [47] with `metric="cosine"` and `min_dist=0.3`. The frequency of predicted query MAIT cells and CD4 CTLs were computed on a per-donor basis and were defined as the frequency given the total number of observed cells per donor. All analyses were run on a computer with one NVIDIA GeForce RTX 3090 GPU.

To evaluate Seurat for this task, we downloaded the processed CITE-seq Seurat object hosted on the Seurat website (https://satijalab.org/seurat/articles/multimodal_reference_mapping.html) and subset it to the same cells used in totalVI. We then applied (using Seurat version 4.0.4) the `ScTransform` normalization to the query data, followed by `FindTransferAnchors` with the `'spca'` reference reduction, followed by `MapQuery`, which produces predicted query labels.

In order to quantify the annotation prediction performance, we first created a bipartite mapping between reference labels and query labels (Supplementary Table 3). After harmonizing these two label sets we then assessed prediction performance using scikit-learn (`sklearn.metrics.f1_score` with `average="weighted"` and `sklearn.metrics.adjusted_mutual_info_score`). Cells that had annotations in the query without a clear reference-level match were excluded from this analysis (Supplementary Table 3). We refer to these scores as "fuzzy F1" and "fuzzy AMI" when comparing labels at the level of the reference to labels at the level of the query.

Supplementary Note 4

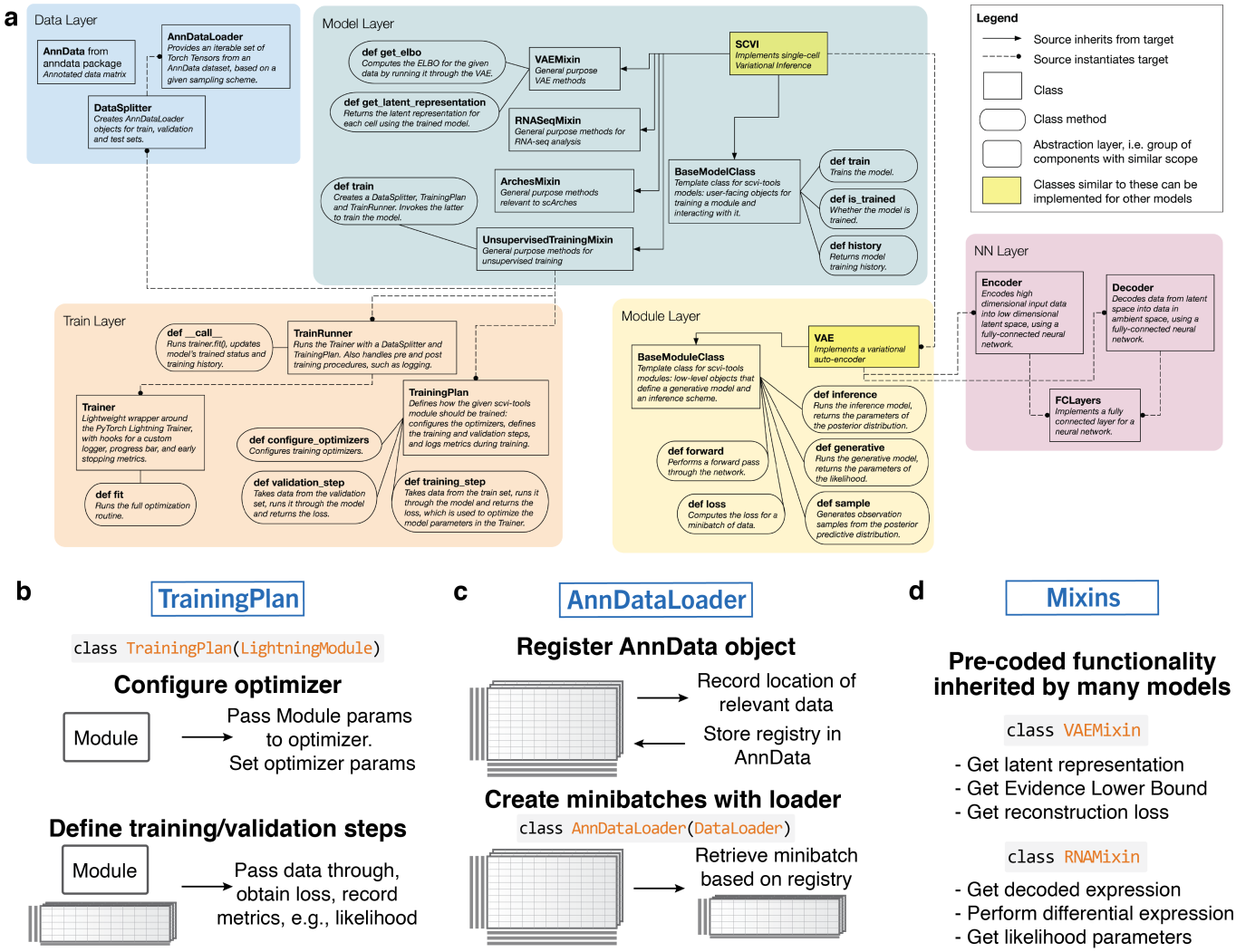
scvi-tools accelerates probabilistic model development

scvi-tools has a convenient programming interface for rapid construction and prototyping of novel probabilistic methods, built on top of PyTorch [48] and AnnData [3]. With this interface, developers can build novel methods through the composition of Python objects (i.e., classes) in scvi-tools that range from being black-box in nature (reused as is) or extensible (can be easily customized) (Supplementary Figure 4a). The primary entry point is the `Model` class, which includes all the components needed to fully specify a new probabilistic model. To ensure flexibility, we implemented the `Model` class in a modular manner through four internally used classes (Figure 2a). The `Module` class specifies the probabilistic form of the model (Figure 2b) and contains the elementary calculations that make up the generative model and the inference procedure, including the objective function to optimize during training (e.g., log likelihood or a lower bound thereof). The `TrainingPlan` class defines the procedure for training the model (Supplementary Figure 4b). This class specifies how to manage stochastic gradient descent in terms of optimizer hyperparameters as well as how to update the parameters of a model given random subsamples (i.e., mini-batches) of data. It also provides an interface with PyTorch Lightning's training procedures [49], which can automatically move the data between different devices such as from CPU to GPU to maximize throughput or perform early stopping. The `AnnDataLoader` class reads data from the AnnData object and automatically structures it for training or for downstream analysis with the trained model (Supplementary Figure 4c). Finally, the `Mixins` are optional classes that implement specific routines for downstream analysis, which can be model-specific or shared among different model classes, such as estimation of differential expression or extraction of latent representations (Supplementary Figure 4d).

These four components may be reused by many models. The `AnnDataLoader` was written as a generic class and already has support for jointly processing data from multiple modalities, such as transcriptomics and proteomics data in totalVI. The `TrainingPlan` subclasses cover a wide range of scenarios, from optimizing a simple objective function, such as in maximum likelihood estimation (MLE), expectation maximization (EM), or variational inference (VI), to more complex semi-supervised learning procedures as is done in scANVI for handling cells with unobserved annotations. Finally, the available `Mixin`(s), like the `VAEMixin` that offers procedures specific to the variational autoencoder (VAE) (listed in Supplementary Figure 4c), can be inherited by new models and augment their functionalities.

Consequently, methods developers can focus on the `Module` class, which specifies the parameters of the model, the metric of fitness (e.g., data likelihood), and (optionally) a recipe for how latent variables can be sampled given the data. The `Module` class has a generic structure consisting of three functions. First, the `generative` function returns the parameters of the data generating distribution, as a function of latent variables, model parameters, and observed covariates. In the case of a VAE (e.g., scVI, totalVI, etc.), the generative function takes samples of the latent variables as input and returns the underlying data distribution encoded according to the generative (decoder) neural network (Figure 2b). In the case of maximum a posteriori (MAP) inference (as in Stereoscope) or EM (as in CellAssign), the `generative` function maps the model parameters to the data generating distribution or returns the components of the expected joint log likelihood, respectively. Second, the `inference` function caters to models that use VI and returns samples from the variational distribution of the latent variables. For a VAE, this calculation is done through an encoder neural network (Figure 2b). Finally, the `loss` function specifies the learning objective given the `inference` and `generative` outputs. For example, the `loss` function can either return the data likelihood (e.g., for MAP), or a lower bound thereof (e.g., for VI or EM).

scvi-tools also contains many other pre-coded building blocks that can be used in the development of new `Module` subclasses for new probablistic models. These include popular neural network architectures, as well as distribution classes that are commonly used for single-cell data, like the mean-parameterized negative binomial. scvi-tools also provides an alternative "backend" with Pyro, which has useful properties like automated `loss` computation and (in some cases) inference procedures like automatic differentiation variational inference (ADVI [50]). We envision that Pyro will be useful for hierarchical Bayesian models with a large collection of latent variables, such as Cell2Location [51]. Pyro will also be appropriate in cases where a black-box inference procedure is known to work well. However, for more complex cases with inference schemes that deviate from standard Bayesian recipes (e.g., warmup [52], alternate variational bounds [53, 54], adversarial



Supplementary Figure 4: Overview of the scvi-tools codebase and exposition of key black-box components. **a**, Static view of the scvi-tools codebase shown at the abstraction level of discrete layers of components. Only select components are shown for each layer and only a single model is presented (scVI). **b**, The training plan configures several aspects relevant for model training, like the optimizers, and the actual training optimization step in which data gets passed through a module. **c**, The `AnnDataLoader` is used to load minibatches of data directly from an AnnData object into a module. **d**, `Mixins` are Python classes with pre-coded functionality that can be inherited into many models, introducing isolated feature sets, like getting metrics relevant to VAEs (VAEMixin).

inference [55]), we recommend using the PyTorch backend. We provide a more technical exposition of these and other capabilities in the tutorials on the scvi-tools website.

AnnData registration and setup in scvi-tools

AnnData objects are the common data structure for single-cell omics data and are the backbone of software like Scanpy [47]. They have also become increasingly easy to convert to and from popular R-based formats like Seurat. Here we provide an overview of how scvi-tools handles AnnData as it relates to the `AnnDataLoader` class.

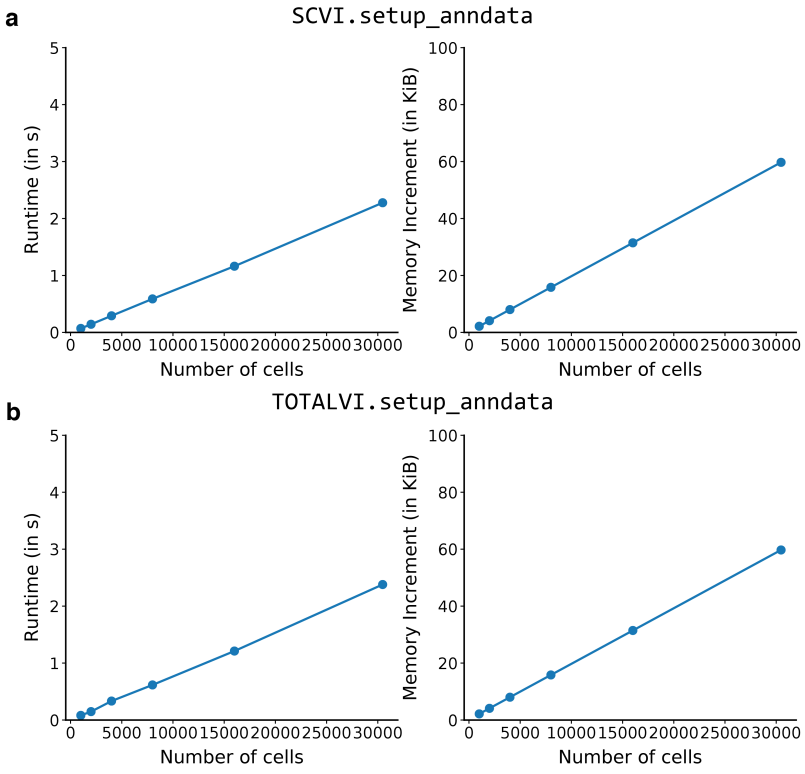
The AnnData object is flexible and allows users to store their data and metadata in locations that are key-valued. Thus, as there is no standard for the names of keys in AnnData, we devised a function in scvi-tools that registers the location of tensors of interest from AnnData such that only these tensors would be loaded into PyTorch-based modules. We named this function `setup_anndata`. We

designed `setup_anndata` to be model-specific, as each particular model may have different data input requirements. For example, scVI can perform integration if a `batch_key` is supplied by the user, but the Stereoscope scRNA-seq model does not perform batch correction and only needs the UMI count data and cell type annotations (through a `labels_key`).

This example is depicted below:

```
import scanpy as sc
from scvi.model import SCVI
from scvi.external.stereoscope import RNAStereoscope

adata = sc.read("my_anndata.h5ad")
SCVI.setup_anndata(adata, batch_key="donor")
model = SCVI(adata)
model.train()

RNAStereoscope.setup_anndata(adata, labels_key="cell_types")
model = RNAStereoscope(adata)
model.train()
```

`setup_anndata` is mandatory to run on an AnnData instance prior to initializing any model. The method itself is very lightweight (Supplementary Figure 5). For each model, `setup_anndata` creates a Python dictionary and adds it back inside the AnnData instance. This dictionary contains information about the shape of the instance (number cells, number genes) as well as registers bookkeeping information, such as the name of each category present at the `batch_key` (if relevant). This is important for validating AnnData objects and preventing unexpected inputs to the model and, in turn, outputs to the user (e.g. if model trained on donors 1-4, but user passes donor 5 after training).



Supplementary Figure 5: Evaluation of memory and runtime of `setup_anndata`. **a**, Plot of runtime and additional memory consumed by the AnnData object after running `SCVI.setup_anndata`. The method was tested against a dataset of CITE-seq measurements from immune cells from murine spleen and lymph node [9]. Before calling `SCVI.setup_anndata`, the dataset was subsetted to 1000, 2000, 4000, 8000, and 16000 cells to test the method's performance at varying dataset sizes. The method was also profiled on the full dataset, comprising of 30474 cells. The method was called with the subsetted AnnData object and the `batch_key` argument, indicating that the model will incorporate prior batch information from the data. The reported runtimes are wall clock time measured by the built-in Python `time` module on a single-core Intel(R) Xeon(R) CPU @ 2.20GHz provided by Google Colaboratory Pro. The memory increments reported were measured as the difference between the results of the Python `sys.getsizeof` method on the AnnData object before and after calling `setup_anndata`. **b**, Plot of runtime and additionally memory consumed by the AnnData object after running `TOTALVI.setup_anndata` with both the `batch_key` argument and the `protein_expression_key` argument. The main difference in this method call involves the incorporation of the protein expression data within the dataset. The other details of the dataset used and the measurements exactly match that of **(a)**.

Supplementary Note 5

Reimplementation of models with scvi-tools

Using the scvi-tools model development interface, we implemented three published methods external to our collaboration: Solo for doublet detection [14], CellAssign for single-cell annotation based on marker genes [13], and Stereoscope for deconvolution of spatial transcriptomics profiles [16]. Additionally, we refactored the codebase for scGen [56, 57], a method for predicting gene expression perturbations on single cells, as well as the codebase for Cell2Location [58], a method for deconvolution of spatial transcriptomics data, to rely on scvi-tools. For all five algorithms, we saw a sharp decrease in number of lines of coded needed. Here we describe the reimplementation of Stereoscope.

Stereoscope Stereoscope is a probabilistic method for deconvolution of spatial transcriptomics profiles, which may represent the average of dozens of cells in each spot [59] (Figure 2c). It is composed of two distinct latent variables models. The first model is trained with an annotated scRNA-seq dataset and learns the gene expression profiles of every annotated cell type. The second model, trained on the spatial data, assumes that the counts in every spot come from a linear combination of the same cell types defined in the scRNA-seq data. The coefficients in this linear combination are normalized and returned as the inferred cell-type proportions at every spot.

There are several reasons for including Stereoscope in scvi-tools. First, while it is a significant and timely contribution for leveraging spatial transcriptomics, it is difficult to use in practice. Indeed, the reference implementation only provides a command line interface to run the algorithm rather than an API, thus complicating its integration in analysis pipelines. Second, Stereoscope is a linear model that is fit with maximum a posteriori inference. It is therefore conceptually different from many of the other models currently implemented in scvi-tools, most of which are deep generative models trained with amortized variational inference. Consequently, this example illustrates the flexibility of our developer interface. A third reason is the elegance and conciseness of this model, which made it a good case study for demonstrating an implementation with our PyTorch backend.

Using the scvi-tools developer interface provided both a conceptual and practical simplification of the reimplementation, focusing most of the effort on the formulation of the actual probabilistic model. Specifically, our reimplementation consisted of two module classes and two model classes (one pair of classes per latent variable model; Figure 2d). It was not necessary to write any code for data loading or training, as these functionalities are inherited through the scvi-tools base classes. Consequently, we observed a marked reduction both in the code complexity (average cyclomatic complexity [60]) and the number of lines of code (Methods) compared to the original codebase (Figure 2e). Our reimplementation also leveraged existing components of scvi-tools. For example, Stereoscope can now use early stopping during training as well as handle data in a sparse format without any additional code due to the common scvi-tools training components. These features, which make the scvi-tools implementation faster and more memory-efficient, are not currently implemented in the original Stereoscope codebase.

To further illustrate the simplicity of implementing new models in scvi-tools, we elaborate on the development of the `ScSignatureModule` class (the module class for the scRNA-seq data latent variable model). In the model, for every cell n, the observed data includes its cell type c_n, its library size l_n and for every gene g, the gene expression x_{ng}. Let G be the number of observed genes and let C be the number of annotated cell types. The parameters of the model, which we want to infer, specify the distribution of each gene g in every cell type c. This distribution is assumed to be a negative binomial, parameterized by λ_{ng} and r_g, where $(r_g\lambda_{ng})/(1-r_g)$ is the expectation and where r_g is a gene-specific parameter determining its mean-variance relationship. Parameter $\lambda_{ng} = l_n\mu_{gc_n}$ of the negative binomial depends on the type assigned to the cell (c_n), its overall number of detected molecules (l_n). Supplementary Figure 6a specifies the model more concisely.

The parameters of the data generative procedure are calculated in `ScSignatureModule` (Supplementary Figure 6b). While the code follows the model closely, care must be given to the constraints on the parameters. For example, μ must be positive, so we use a softplus transformation. Similarly, r must be in the range $[0, 1]$, which we enforce with a sigmoid transformation. The loss function returns the likelihood of the observed data, using a negative binomial distribution and evaluated at the model parameters. In contrast to VAEs (Figure 2b), there is no need to provide an implementation of the inference method because there are no latent variables. Our implementation therefore consists only of

the generative function and the loss.

a **Mathematical description**

Parameters, on appropriate logit-scale:

μ_{gc} cell-type specific gene expression
r_g gene specific success rate

Data:

x_{ng} gene expression
l_n library size
c_n cell type

Data generating process:

$$x_{ng} \mid c_n \sim \text{NegativeBinomial}\left(l_n \mu_{gc_n}, r_g\right)$$

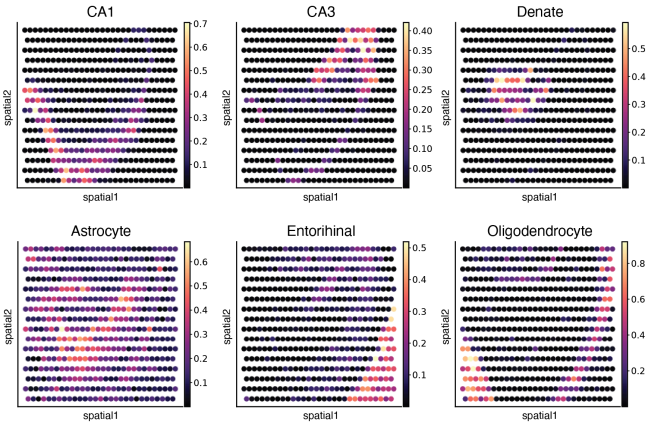
b **Module pseudocode**

```
class ScSignatureModule(BaseModuleClass):


    def generative(self, x, c):
        # mapping data to parameters of the likelihood
        l = x.sum(dim=1)
        px_lambda = l * softplus(μ)[:, c]
        px_rate = sigmoid(r)
        dist = NegativeBinomial(px_lambda, px_rate)
        return dist

    def loss(self, x, dist):
        # computing the data likelihood
        return -dist.log_prob(x).sum(-1).mean()
```

c **User experience**

```
import scvi
import scanpy as sc
# load the RNA data, learn cell-type signatures
sc_adata = scvi.data.read_h5ad("hippocampus_scRNA.h5ad")
sc_model = ScStereoscope(sc_adata)
sc_model.train()

# deconvolution of spatial data using signatures
st_adata = scvi.data.read_h5ad("hippocampus_spatial.h5ad")
st_model = SpatialStereoscope.load_rna_model(
    st_adata=st_adata,
    sc_model=sc_model,
)
st_model.train()
# store signatures in anndata, visualize with scanpy
st_adata.obsm["proportions"] = st_model.get_proportions()
sc.pl.embedding(st_adata, basis="spatial", color=cell_types)
```

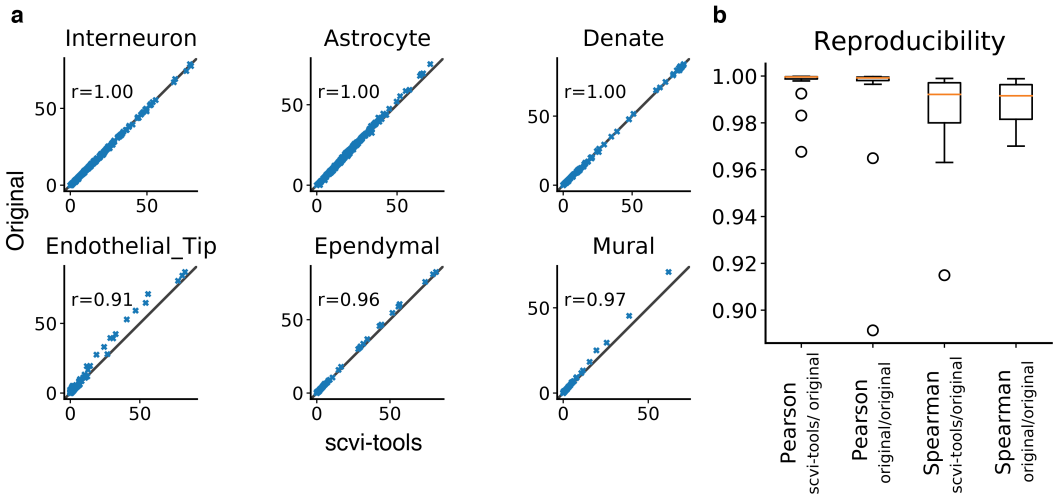
d



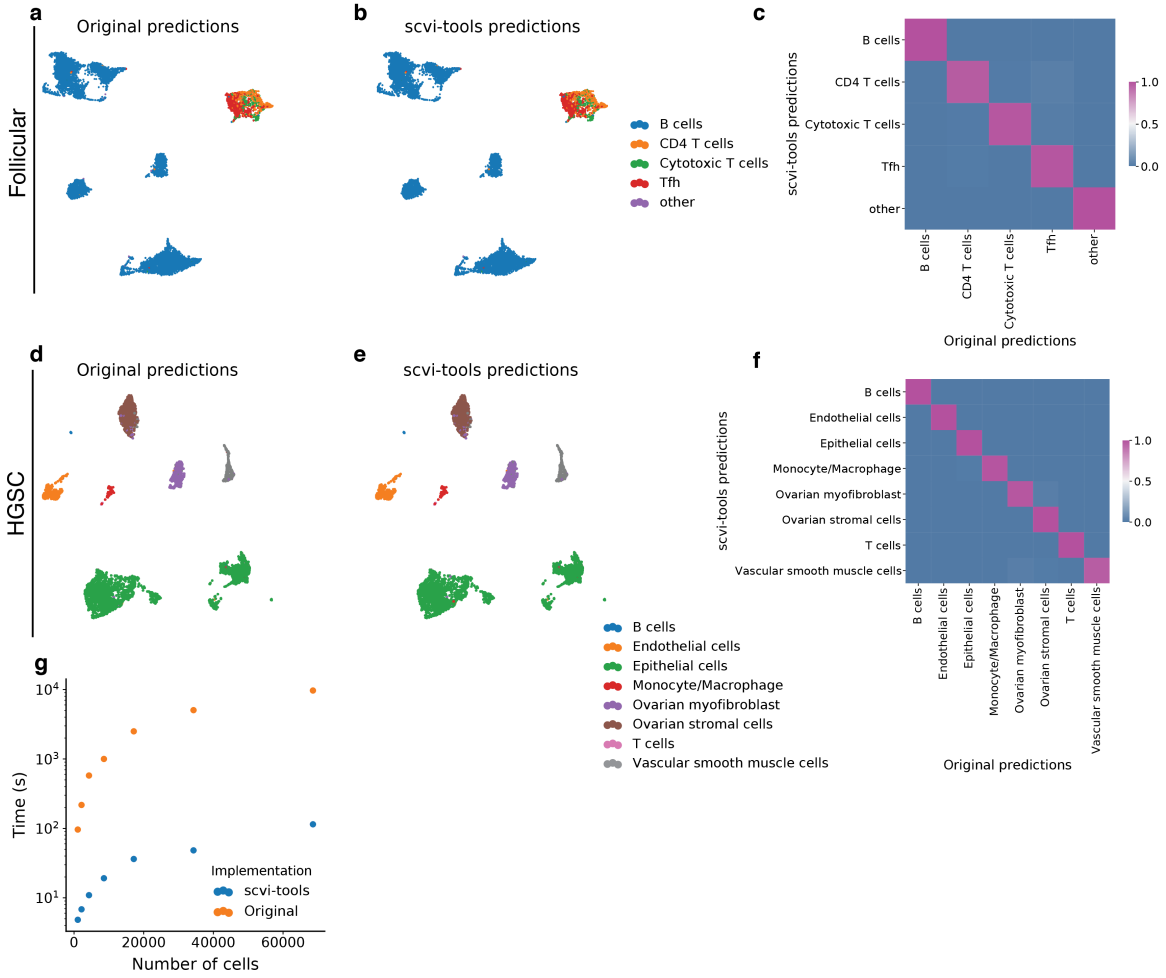
Supplementary Figure 6: Reimplementation of Stereoscope in scvi-tools. a, b, Description of implementation of the `ScSignatureModule`, the module class for the single-cell model of the Stereoscope method. **c**, Example of user (pseudo)code and interaction with Scanpy. **d**, Output example on the hippocampus spatial 10x Visium dataset.

We applied the method to the 10x Visium spatial transcriptomics data of an adult mouse brain [61] and a single-cell RNA sequencing dataset of the mouse hippocampus [62] (Methods). Applying Stereoscope and visualizing the results with Scanpy takes less than 20 lines of code including the import statements and the call to the Scanpy library (Supplementary Figure 6c). In contrast to the original software, which only accommodated a command line interface, our reimplementation can be used from Jupyter notebooks and with AnnData objects. The result of our Stereoscope implementation (Supplementary Figure 6d) performs nearly the same as the original implementation (Supplementary Figure 7) in terms of the average Spearman correlation of cell-type proportion across all individual cell types. This is expected as the module classes of the reimplementation are a reorganization of their analogs in the original codebase.

CellAssign CellAssign is a model for automated annotation of scRNA-seq data based on overexpression of markers. The implementation provides annotations that are reproducible with the original implementation with an increase in inference time due to minibatching (Methods). An overview of the scvi-tools implementation of CellAssign can be found in the scvi-tools online user guide https://docs.scvi-tools.org/en/stable/user_guide/models/cellassign.html.



Supplementary Figure 7: Reproducibility of Stereoscope implementation in scvi-tools. a, Scatter plot for six cell types of the hippocampus dataset. Each point in the scatter plot represents one spot. The x axis is the unnormalized proportion inferred by the original Stereoscope software and the y axis is the unnormalized proportion inferred by our implementation. We report the Spearman coefficient for each cell type. The top three cell types are the most reproducible and the bottom three cell types are the least reproducible. **b**, Box plot of correlations of unnormalized proportions between the two implementations. Box plots were computed on n=17 cell types and indicate the median (center lines), interquartile range (hinges), and whiskers at 1.5× interquartile range.



Supplementary Figure 8: Evaluation of CellAssign implementation in scvi-tools. **a**, **b**, UMAP embedding of follicular lymphoma single-cell expression data, labeled by maximum probability assignments from the **(a)** original CellAssign implementation and **(b)** scvi-tools implementation. **c**, Row-normalized confusion matrix of scvi-tools predicted labels (rows) and study-derived cell annotations (columns) for follicular lymphoma expression data. **d**, **e**, UMAP embedding of HGSC single-cell expression data, labeled by maximum probability assignments from the **(d)** original CellAssign implementation and **(e)** scvi-tools implementation. **f**, Row-normalized confusion matrix of scvi-tools predicted labels (rows) and study-derived cell annotations (columns) for HGSC single-cell expression data. **g**, Runtime on downsampled versions of 68k peripheral blood mononuclear cells dataset for the original and scvi-tools implementations.

Methods

Stereoscope For the single-cell data, we used the dataset from Saunders et al. [62], as pre-processed by Cable et al [63]. We filtered genes out with a minimum count of 10. Then, we selected 2,000 highly variable genes using the corresponding scanpy function. For the spatial transcriptomics data, we used the V1 Adult Mouse Brain dataset [61] and filtered spots so as to focus on the hippocampus (as in Cable et al. Supplementary Figure 7). We then filtered genes in the spatial transcriptomics data by taking the intersection with the highly variable genes in the single-cell data. We then ran the single-cell model for 100 epochs and ran the spatial model for 5,000 epochs. We ran the original Stereoscope code from the command line, on the same dataset and with the same parameters. We used the Radon package for the calculations of average cyclomatic complexity [60] and source lines of code.

CellAssign We downloaded two datasets with full metadata and cell-type marker matrices from the original publication [13] (from `https://zenodo.org/record/3372746`) and compared the scvi-tools implementation predictions to the original predictions for these datasets. The first dataset consisted of 9,156 cells from lymph node biopses of two follicular lymphoma (FL) patients. The second dataset consisted of 4,848 cells from a high-grade serous carcinoma (HGSC) patient. On both datasets, the scvi-tools implementation predictions were highly reproducible with the original implementation (Supplementary Figure 8a-f). UMAP embeddings used in Supplementary Figure 8 were the original embeddings from the publication and were retrieved from the downloaded objects.

Next, we evaluated the runtime of the two implementations on a desktop with an Intel Core i9-10900K 3.7 GHz processor, 2x CorsairVengeance LPX 64GB ram, and an NVIDIA RTX 3090 GPU. To do so, we used a dataset of 68k peripheral blood mononuclear cells from 10x Genomics [61], and used the same cell-type marker matrix as the FL dataset (23 markers, 5 cell types). Across a range of cells, we observed that the scvi-tools implementation is consistently faster and thus more scalable (Supplementary Figure 8g). This is mostly due in part to the minibatching, which is a capability present in the original codebase, but not set as default (also no guidance on how to set it). Thus, we have shown that the scvi-tools implementation of CellAssign is both reproducible, and by default, more scalable than the original implementation.

Supplementary Tables

Modality	Model	Tasks
scRNA-seq	scVI [6]	Dimensionality reduction, removal of unwanted variation, integration across replicates, donors, and technologies, differential expression, imputation, normalization of other cell- and sample-level confounding factors
	LDVAE [11]	scVI tasks with linear decoder
	scANVI [8]	Automated annotation, all scVI tasks
	AutoZI [64]	Gene-wise model selection for zero-inflation
	Solo [14]	Doublet detection
	CellAssign [13]	Marker-based automated annotation
CITE-seq	totalVI [9]	Dimensionality reduction, removal of unwanted variation, integration across replicates, donors, and technologies, differential expression, protein imputation, imputation, normalization of other cell- and sample-level confounding factors
scATAC-seq	PeakVI [10]	Dimensionality reduction, removal of unwanted variation, differential accessibility, imputation, normalization of other cell- and sample-level confounding factors
Spatial Transcriptomics	gimVI [19]	Imputation of missing genes in spatial data using scRNA-seq reference
	Stereoscope [16]	Deconvolution of spatial transcriptomics profiles
	DestVI [17]	Multi-resolution deconvolution of spatial transcriptomics profiles
Multiple	scArches [38]	Transfer learning for reference-query integration applied on top of peakVI, scVI, scANVI, totalVI
	MultiVI [12]	Integration of paired/unpaired multiome data, missing modality imputation, normalization of other cell- and sample-level confounding factors
	LDA [15]	Topic modeling, dimensionality reduction

Supplementary Table 1: Overview of models and functionality currently implemented in scvi-tools. A web-based table is accessible at `https://docs.scvi-tools.org/en/stable/user_guide/index.html`

Cell Sex Genes	Cell Cycle Genes	Marker Genes
lncRNA:roX1	PCNA	Argk
lncRNA:roX2	dnk	Nrt
Sxl	RnrS	Ten-a
msl-2	RnrL	Ten-m
	Claspin	wb
	Mcm5	Act57B
	Caf1-180	drl
	RPA2	mid
	HipHop	nemy
	stg	lms
	Mcm6	CG11835
	dup	Gyg
	WRNexo	ara
	Mcm7	tok
	dpa	kirre
	CG10336	NK7.1
	Mcm3	fj
	Mcm2	beat-IIIc
	RpA-70	CG33993
	Chrac-14	dpr16
	CG13690	CG15529
	RPA3	CG9593
	asf1	beat-IIb
	DNApol-alpha73	robo2
	CycE	Ama
	DNApol-alpha50	fz2
	Kmn1	elB
	Lam	noc
	Nph	nkd
	msd5	fng
	msd1	vg
	ctp	
	Set	
	scra	
	Chrac-16	
	ncd	
	Ote	
	pzg	
	HDAC1	
	nesd	
	tum	
	CG8173	
	aurB	
	feo	
	pav	
	CG6767	
	sip2	
	Det	
	Cks30A	
	CycB	
	B52	

Supplementary Table 2: Gene sets for multiple covariates Drosophila analysis.

Joint label	Reference label	Query label
0	B intermediate, B naive, B memory	C7-B
1	CD14 Mono	C3-C MONO_1, C16-C MONO_2, C22-C MONO_3, C11-C MONO_IFN
2	cDC1	C24-CDC1
3	cDC2	C5-CDC2
4	pDC	C10-PDC
5	CD4 Naive, CD4 TCM, CD4 TEM, CD4 Proliferating, Treg	C0-CD4
6	CD8 Naive, CD8 TCM, CD8 TEM, CD8 Proliferating	CD-CD8
7	NK, NK Proliferating, NK_CD56bright	CD-NK
8	Platelet	C6-PLATE_1, C14-PLATE_2
9	Plasmablast	C13-PB_1, C15-PB_2
10	HSPC	C19-HSC
11	Eryth	C17-RBC
N/A	CD4 CTL, dnT, MAIT, gdT, ILC, ASDC	C9-GRAN, C18-T_IFN, C23-T, C20-BASO, C21-Cl21, C12-EOS

Supplementary Table 3: Mapping of reference and query labels.

Data availability

A collection of processed data discussed in this manuscript have been deposited on figshare (https://doi.org/10.6084/m9.figshare.14374574.v1).

Code availability

The code to reproduce the experiments of this manuscript is available at https://github.com/YosefLab/scvi-tools-reproducibility. The `scvi-tools` package can be found on GitHub at https://github.com/YosefLab/scvi-tools, and is also deposited on Zenodo https://doi.org/10.5281/zenodo.4341715. Documentation and tutorials can be found at https://scvi-tools.org.

References

[1] Tallulah S Andrews, Vladimir Yu Kiselev, Davis McCarthy, and Martin Hemberg. "Tutorial: guidelines for the computational analysis of single-cell RNA sequencing data". In: *Nature Protocols* (2020).

[2] Malte D Luecken and Fabian J Theis. "Current best practices in single-cell RNA-seq analysis: a tutorial". In: *Molecular Systems Biology* (2019).

[3] F Alexander Wolf, Philipp Angerer, and Fabian J Theis. "SCANPY: large-scale single-cell gene expression data analysis". In: *Genome Biology* (2018).

[4] Rahul Satija, Jeffrey A Farrell, David Gennert, Alexander F Schier, and Aviv Regev. "Spatial reconstruction of single-cell gene expression data". In: *Nature Biotechnology* (2015).

[5] Davis J McCarthy, Kieran R Campbell, Aaron T L Lun, and Quin F Wills. "Scater: pre-processing, quality control, normalization and visualization of single-cell RNA-seq data in R". In: *Bioinformatics* (2017).

[6] Romain Lopez, Jeffrey Regier, Michael B Cole, Michael I Jordan, and Nir Yosef. "Deep generative modeling for single-cell transcriptomics". In: *Nature Methods* (2018).

[7] Pierre Boyeau, Romain Lopez, Jeffrey Regier, Adam Gayoso, Michael I Jordan, and Nir Yosef. "Deep generative models for detecting differential expression in single cells". In: *bioRxiv* (2019).

[8] Chenling Xu, Romain Lopez, Edouard Mehlman, Jeffrey Regier, Michael I. Jordan, and Nir Yosef. "Probabilistic harmonization and annotation of single-cell transcriptomics data with deep generative models". In: *Molecular Systems Biology* (2021).

[9] Adam Gayoso, Zoë Steier, Romain Lopez, Jeffrey Regier, Kristopher L Nazor, Aaron Streets, and Nir Yosef. "Joint probabilistic modeling of single-cell multi-omic data with totalVI". In: *Nature Methods* (2021).

[10] Tal Ashuach, Daniel A. Reidenbach, Adam Gayoso, and Nir Yosef. "PeakVI: A Deep Generative Model for Single Cell Chromatin Accessibility Analysis". In: *bioRxiv* (2021).

[11] Valentine Svensson, Adam Gayoso, Nir Yosef, and Lior Pachter. "Interpretable factor models of single-cell RNA-seq via variational autoencoders". In: *Bioinformatics* (2020).

[12] Tal Ashuach, Mariano I Gabitto, Michael I Jordan, and Nir Yosef. "MultiVI: deep generative model for the integration of multi-modal data". In: *bioRxiv* (2021).

[13] Allen W Zhang, Ciara O'Flanagan, Elizabeth A Chavez, Jamie LP Lim, Nicholas Ceglia, Andrew McPherson, Matt Wiens, Pascale Walters, Tim Chan, et al. "Probabilistic cell-type assignment of single-cell RNA-seq for tumor microenvironment profiling". In: *Nature Methods* (2019).

[14] Nicholas J Bernstein, Nicole L Fong, Irene Lam, Margaret A Roy, David G Hendrickson, and David R Kelley. "Solo: doublet identification in single-cell RNA-Seq via semi-supervised deep learning". In: *Cell Systems* (2020).

[15] David M Blei, Andrew Y Ng, and Michael I Jordan. "Latent dirichlet allocation". In: *the Journal of machine Learning research* (2003).

[16] Alma Andersson, Joseph Bergenstråhle, Michaela Asp, Ludvig Bergenstråhle, Aleksandra Jurek, José Fernández Navarro, and Joakim Lundeberg. "Single-cell and spatial transcriptomics enables probabilistic inference of cell type topography". In: *Communications Biology* (2020).

[17] Romain Lopez, Baoguo Li, Hadas Keren-Shaul, Pierre Boyeau, Merav Kedmi, David Pilzer, Adam Jelinski, Eyal David, Allon Wagner, et al. "Multi-resolution deconvolution of spatial transcriptomics data reveals continuous patterns of inflammation". In: *bioRxiv* (2021).

[18] Marlon Stoeckius, Christoph Hafemeister, William Stephenson, Brian Houck-Loomis, Pratip K Chattopadhyay, Harold Swerdlow, Rahul Satija, and Peter Smibert. "Simultaneous epitope and transcriptome measurement in single cells". In: *Nature Methods* (2017).

[19] Romain Lopez, Achille Nazaret, Maxime Langevin, Jules Samaran, Jeffrey Regier, Michael I Jordan, and Nir Yosef. "A joint model of unpaired data from scRNA-seq and spatial transcriptomics for imputing missing gene expression measurements". In: *ICML workshop in Computational Biology*. 2019.

[20] David S Fischer, Leander Dony, Martin König, Abdul Moeed, Luke Zappia, Lukas Heumos, Sophie Tritschler, Olle Holmberg, Hananeh Aliee, et al. "Sfaira accelerates data and model reuse in single cell genomics". In: *Genome Biology* (2021).

[21] JJ Allaire, Kevin Ushey, Yuan Tang, and Dirk Eddelbuettel. *reticulate: R Interface to Python*. 2017. URL: https://github.com/rstudio/reticulate.

[22] David DeTomaso, Matthew G Jones, Meena Subramaniam, Tal Ashuach, J Ye Chun, and Nir Yosef. "Functional interpretation of single cell similarity maps". In: *Nature Communications* (2019).

[23] Colin Megill, Bruce Martin, Charlotte Weaver, Sidney Bell, Lia Prins, Seve Badajoz, Brian McCandless, Angela Oliveira Pisco, Marcus Kinsella, et al. "cellxgene: a performant, scalable exploration platform for high dimensional sparse matrices". In: *bioRxiv* (2021).

[24] Johann A. Gagnon-Bartsch, Laurent Jacob, and Terence P. Speed. *Removing unwanted variation from high dimensional data with negative controls*. Tech. rep. 2013. URL: https://statistics.berkeley.edu/sites/default/files/tech-reports/ruv.pdf.

[25] Andrew Butler, Paul Hoffman, Peter Smibert, Efthymia Papalexi, and Rahul Satija. "Integrating single-cell transcriptomic data across different conditions, technologies, and species". In: *Nature biotechnology* (2018).

[26] Leland McInnes, John Healy, and James Melville. "UMAP: Uniform manifold approximation and projection for dimension reduction". In: *Journal of Open Source Software* (2018).

[27] Malte D Luecken, Maren Buttner, Kridsadakorn Chaichoompu, Anna Danese, Marta Interlandi, Michaela F Müller, Daniel C Strobl, Luke Zappia, Martin Dugas, et al. "Benchmarking atlas-level data integration in single-cell genomics". In: *bioRxiv* (2020).

[28] Hoa Thi Nhu Tran, Kok Siong Ang, Marion Chevrier, Xiaomeng Zhang, Nicole Yee Shin Lee, Michelle Goh, and Jinmiao Chen. "A benchmark of batch-effect correction methods for single-cell RNA sequencing data". In: *Genome Biology* (2020).

[29] Ilya Korsunsky, Nghia Millard, Jean Fan, Kamil Slowikowski, Fan Zhang, Kevin Wei, Yuriy Baglaenko, Michael Brenner, Po-ru Loh, et al. "Fast, sensitive and accurate integration of single-cell data with Harmony". In: *Nature Methods* (2019).

[30] W. Evan Johnson, Cheng Li, and Ariel Rabinovic. "Adjusting batch effects in microarray expression data using empirical Bayes methods". In: *Biostatistics* (2006).

[31] Nicholas J. Everetts, Melanie I. Worley, Riku Yasutomi, Nir Yosef, and Iswar K. Hariharan. "Single-cell transcriptomics of the Drosophila wing disc reveals instructive epithelium-to-myoblast interactions". In: *eLife* (2021).

[32] Yuhan Hao, Stephanie Hao, Erica Andersen-Nissen, William M Mauck III, Shiwei Zheng, Andrew Butler, Maddie J Lee, Aaron J Wilk, Charlotte Darby, et al. "Integrated analysis of multimodal single-cell data". In: *Cell* (2021).

[33] Krzysztof Polański, Matthew D Young, Zhichao Miao, Kerstin B Meyer, Sarah A Teichmann, and Jong-Eun Park. "BBKNN: Fast batch alignment of single cell transcriptomes". In: *Bioinformatics* (2019).

[34] R. C. Geary. "The contiguity ratio and statistical mapping". In: *The Incorporated Statistician* (1954).

[35] Monika Litviňuková, Carlos Talavera-López, Henrike Maatz, Daniel Reichart, Catherine L Worth, Eric L Lindberg, Masatoshi Kanda, Krzysztof Polanski, Matthias Heinig, et al. "Cells of the adult human heart". In: *Nature* (2020).

[36] Maria Paula Zappia, Lucia de Castro, Majd M Ariss, Holly Jefferson, Abul BMMK Islam, and Maxim V Frolov. "A cell atlas of adult muscle precursors uncovers early events in fibre-type divergence in Drosophila". In: *EMBO reports* (2020).

[37] Vinay S Swamy, Temesgen D Fufa, Robert B Hufnagel, and David M McGaughey. "Building the mega single cell transcriptome ocular meta-atlas". In: *bioRxiv* (2021).

[38] Mohammad Lotfollahi, Mohsen Naghipourfar, Malte Luecken, Matin Khajavi, Maren Büttner, Ziga Avsec, Alexander Misharin, and Fabian Theis. "Query to reference single-cell integration with transfer learning". In: *bioRxiv* (2020).

[39] Tim Stuart, Andrew Butler, Paul Hoffman, Christoph Hafemeister, Efthymia Papalexi, William M Mauck III, Yuhan Hao, Marlon Stoeckius, Peter Smibert, et al. "Comprehensive integration of single-cell data". In: *Cell* (2019).

[40] Prabhu S. Arunachalam, Florian Wimmers, Chris Ka Pun Mok, Ranawaka A.P.M. Perera, Madeleine Scott, Thomas Hagan, Natalia Sigal, Yupeng Feng, Laurel Bristow, et al. "Systems biological assessment of immunity to mild versus severe COVID-19 infection in humans". In: *Science* (2020).

[41] Els Wauters, Pierre Van Mol, Abhishek Dinkarnath Garg, Sander Jansen, Yannick Van Herck, Lore Vanderbeke, Ayse Bassez, Bram Boeckx, Bert Malengier-Devlies, et al. "Discriminating mild from critical COVID-19 by innate and adaptive immune single-cell profiling of bronchoalveolar lavages". In: *Cell Research* (2021).

[42] Tiphaine Parrot, Jean Baptiste Gorin, Andrea Ponzetta, Kimia T. Maleki, Tobias Kammann, Johanna Emgård, André Perez-Potti, Takuya Sekine, Olga Rivera-Ballesteros, et al. "MAIT cell activation and dynamics associated with COVID-19 disease severity and outcome". In: *Science Immunology* (2020).

[43] Jennifer A. Juno, David van Bockel, Stephen J. Kent, Anthony D. Kelleher, John J. Zaunders, and C. Mee Ling Munier. "Cytotoxic CD4 T cells-friend or foe during viral infection?" In: *Frontiers in Immunology* (2017).

[44] Benjamin J. Meckiff, Ciro Ramírez-Suástegui, Vicente Fajardo, Serena J. Chee, Anthony Kusnadi, Hayley Simon, Alba Grifoni, Emanuela Pelosi, Daniela Weiskopf, et al. "Single-cell transcriptomic analysis of SARS-CoV-2 reactive CD4 + T cells". In: *bioRxiv* (2020).

[45] Samuel L Wolock, Romain Lopez, and Allon M Klein. "Scrublet: computational identification of cell doublets in single-cell transcriptomic data". In: *Cell Systems* (2019).

[46] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, et al. "Scikit-learn: Machine learning in Python". In: *Journal of Machine Learning Research* (2011).

[47] F. Alexander Wolf, Philipp Angerer, and Fabian J. Theis. "SCANPY: Large-scale single-cell gene expression data analysis". In: *Genome Biology* (2018).

[48] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, et al. "Automatic differentiation in PyTorch". In: *NIPS Workshop Autodiff*. 2017.

[49] WA Falcon and .al. "PyTorch Lightning". In: *GitHub* (2019). URL: https://github.com/PyTorchLightning/pytorch-lightning.

[50] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. "Automatic differentiation variational inference". In: *The Journal of Machine Learning Research* (2017).

[51] Vitalii Kleshchevnikov, Artem Shmatko, Emma Dann, Alexander Aivazidis, Hamish W King, Tong Li, Artem Lomakin, Veronika Kedlian, Mika Sarkin Jain, et al. "Comprehensive mapping of tissue cell architecture via integrated single cell and spatial transcriptomics". In: *bioRxiv* (2020).

[52] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. "Ladder variational autoencoders". In: *Advances in Neural Information Processing Systems* (2016).

[53] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. "Importance weighted autoencoders". In: *International Conference on Learning Representations*. 2016.

[54] Romain Lopez, Pierre Boyeau, Nir Yosef, Michael I. Jordan, and Jeffrey Regier. "Decision-making with auto-encoding variational Bayes". In: *Advances in Neural Information Processing Systems* (2020).

[55] Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. "Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks". In: *International Conference on Machine Learning*. PMLR. 2017.

[56] Mohammad Lotfollahi, F Alexander Wolf, and Fabian J Theis. "scGen predicts single-cell perturbation responses". In: *Nature Methods* (2019).

[57] Mohammad Lotfollahi. *scGen codebase*. 2021. URL: https://github.com/theislab/scgen.

[58] Vitalii Kleshchevnikov, Artem Shmatko, Emma Dann, Alexander Aivazidis, Hamish W King, Tong Li, Artem Lomakin, Veronika Kedlian, Mika Sarkin Jain, et al. "Comprehensive mapping of tissue cell architecture via integrated single cell and spatial transcriptomics". In: *bioRxiv* (2020).

[59] Patrik L. Ståhl, Fredrik Salmén, Sanja Vickovic, Anna Lundmark, José Fernández Navarro, Jens Magnusson, Stefania Giacomello, Michaela Asp, Jakub O. Westholm, et al. "Visualization and analysis of gene expression in tissue sections by spatial transcriptomics". In: *Science* (2016).

[60] Thomas J McCabe. "A complexity measure". In: *IEEE Transactions on software Engineering* (1976).

[61] *10x Genomics*. 2017. URL: https://support.10xgenomics.com/single-cell-gene-expression/datasets.

[62] Arpiar Saunders, Evan Z Macosko, Alec Wysoker, Melissa Goldman, Fenna M Krienen, Heather de Rivera, Elizabeth Bien, Matthew Baum, Laura Bortolin, et al. "Molecular diversity and specializations among the cells of the adult mouse brain". In: *Cell* (2018).

[63] Dylan M Cable, Evan Murray, Luli S Zou, Aleksandrina Goeva, Evan Z Macosko, Fei Chen, and Rafael A Irizarry. "Robust decomposition of cell type mixtures in spatial transcriptomics". In: *Nature Biotechnology* (2021).

[64] Oscar Clivio, Romain Lopez, Jeffrey Regier, Adam Gayoso, Michael I Jordan, and Nir Yosef. "Detecting zero-inflated genes in single-cell transcriptomics data". In: *Machine Learning in Computational Biology (MLCB)*. 2019.