

Bonsai reconstructs tree representations for distortion-free visualization and exploration of high-dimensional data

In the format provided by the
authors and unedited

Supplementary Information for: *Bonsai* reconstructs tree representations for distortion-free visualization and exploration of high-dimensional data

Daan H. de Groot^{1✉}, Sarah X. Morillo Leonardo¹, Mikhail Pachkov¹, and Erik van Nimwegen^{1✉}

¹Biozentrum, University of Basel, and Swiss Institute of Bioinformatics, Basel, Switzerland

✉email: daanhugodegroot@gmail.com; erik.vannimwegen@unibas.ch

Table of Contents

SI.A	Supplementary Figures	3
SI.B	The likelihood of the data given a tree	37
SI.B.1	The likelihood of a tree with given branch lengths	37
SI.B.1.1	Likelihood of the data at the leaf-nodes given a true position	38
SI.B.1.2	The likelihood of gene expression data given a cell's true gene expression state	38
SI.B.1.3	Selecting features with signal-to-noise over a threshold	40
SI.B.1.4	The prior probability of movement in gene expression state	41
SI.B.1.5	The full tree likelihood expression	42
SI.B.1.6	The tree likelihood for other types of data	42
SI.B.2	Analytical solutions for the tree likelihood	43
SI.B.2.1	Solving the integral for the star-tree	44
SI.B.2.2	Summarizing a sub-tree in an <i>effective leaf</i>	45
SI.B.2.3	The loglikelihood of a general tree.	47
SI.B.3	Optimization of the branch lengths	49
SI.B.4	Optimally placing a new cell on an existing tree	49
SI.B.4.1	Strategies for finding the optimal attachment node	50
SI.B.4.2	Attaching the new cell to an edge	51
SI.C	Finding the tree with maximum posterior	51
SI.C.1	Start from a star-tree	51
SI.C.2	Iteratively add the most likely internal node	52
SI.C.2.1	Finding the ancestor that increases the likelihood most	52
SI.C.3	Further resolve polytomies	54
SI.C.4	Optimize branch lengths	54
SI.C.5	Subtree pruning and regrafting	55
SI.C.5.1	Pruning a subtree	55
SI.C.5.2	Regrafting the subtree	55
SI.C.6	Interchange nearest neighbors	55
SI.C.7	Optimize branch lengths a final time	56

SI.D	Speeding up the tree reconstruction	56
SI.D.1	Calculating upper bounds for merging likelihoods given a change in the root's position	56
SI.D.1.1	Dynamically setting the ellipsoid size	60
SI.D.2	Limiting the number of candidate-pairs	60
SI.D.3	The effect on <i>Bonsai</i> 's computation time	61
SI.E	Creating simulated scRNA-seq datasets	61
SI.E.1	Sampling mRNA-counts given LTQ-values	61
SI.E.2	Simulating LTQ-values for each cell	62
SI.E.2.1	A binary tree with constant branch lengths	62
SI.E.2.2	Varying branch lengths	63
SI.E.2.3	An unbalanced tree	63
SI.E.2.4	A tree embedded in a lower-dimensional subspace	63
SI.E.2.5	A pseudo-bulk-based dataset	64
SI.F	Creating more general simulated datasets	64
SI.F.1	Noise-free datasets with increasing dimensionality	64
SI.F.2	Noisy datasets with an increasing number of objects	65
SI.G	Tree visualization	65
SI.G.1	Tree layout algorithms	65
SI.G.2	The interpretation of tree distances	66
SI.G.2.1	Distances in the dendrogram layout	66
SI.G.2.2	Distances on a disk with hyperbolic geometry	66
SI.H	Analysis of football player statistics	67

SI.A Supplementary Figures

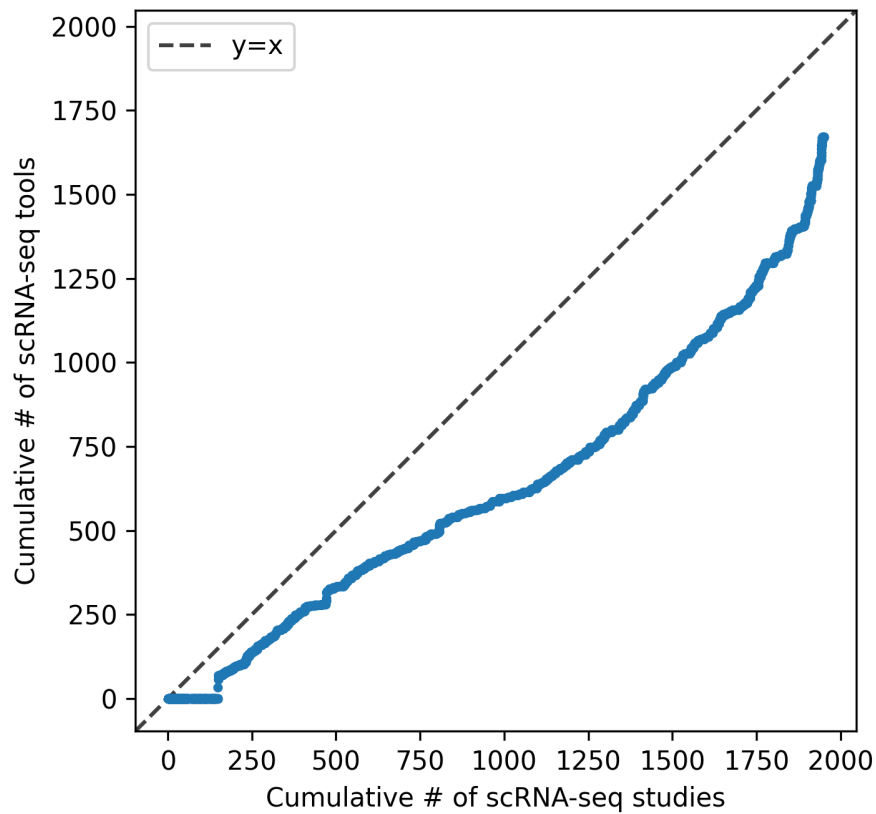


Figure S1. The increase in the number of published scRNA-seq tools almost keeps up with the number of published scRNA-seq datasets. This plot was created using code adapted from https://pachterlab.github.io/kallistobustools/tutorials/scRNA-seq_intro/python/scRNA-seq_intro/ and using data from [1] and [2]. The black diagonal line indicates $y = x$.

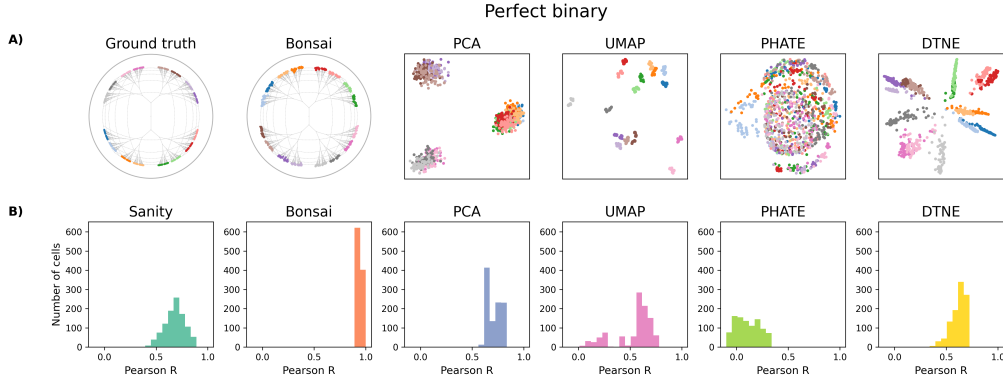


Figure S2. Visualization and pairwise distance recovery on a simulated dataset based on a perfectly binary tree. **A)** Ground truth tree (left, see section S1.E for information on the simulation) and visualizations by *Bonsai*, PCA, UMAP, PHATE, and DTNE. The *Bonsai* visualization is shown in the equal-daylight layout on the hyperbolic disk. For PCA, UMAP, PHATE, and DTNE, the data are preprocessed with a standard log1p-normalization after which a PCA-projection is performed. To present these methods in the best light, we performed a coarse parameter scan and for each method used the number of PCA components that performed best. This resulted in keeping all dimensions for UMAP, projecting to 500 PCs for DTNE, and 100 PCs for PHATE. The UMAP-embedding into two dimensions was run with parameters: `n_neighbors=15`, `min_dist=0.1`, `n_components=2`, `metric='euclidean'`, and we used the default parameters for DTNE and PHATE. **B)** Histograms of, for each method, the correlations (Pearson R) between the ground truth distances from each cell to all other cells and the corresponding distances in the visualization. For *Sanity*, we used the specialized distance-calculation program, see [3].

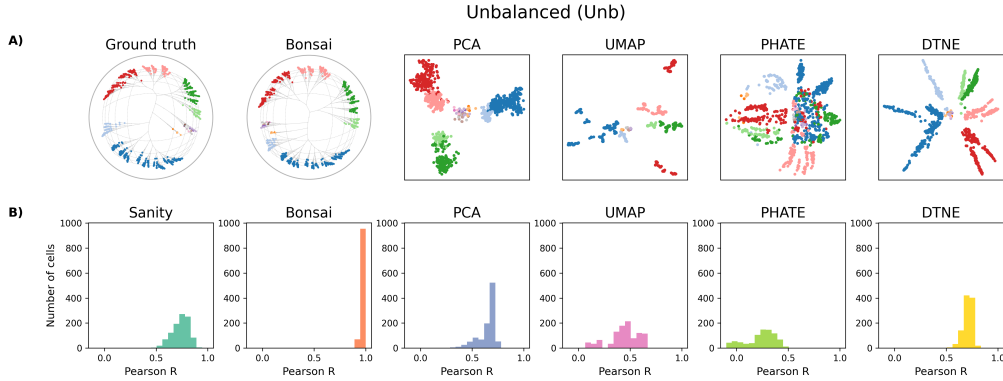


Figure S3. Visualization and pairwise distance recovery on a simulated dataset based on an unbalanced tree. **A)** Ground truth (left) and visualizations by different methods of simulated data of an unbalanced tree (see Section S1.E.2 for information on the simulation). **B)** Histograms of, for each method, the correlations (Pearson R) between the ground truth distances from each cell to all other cells and the corresponding distances in the visualization.

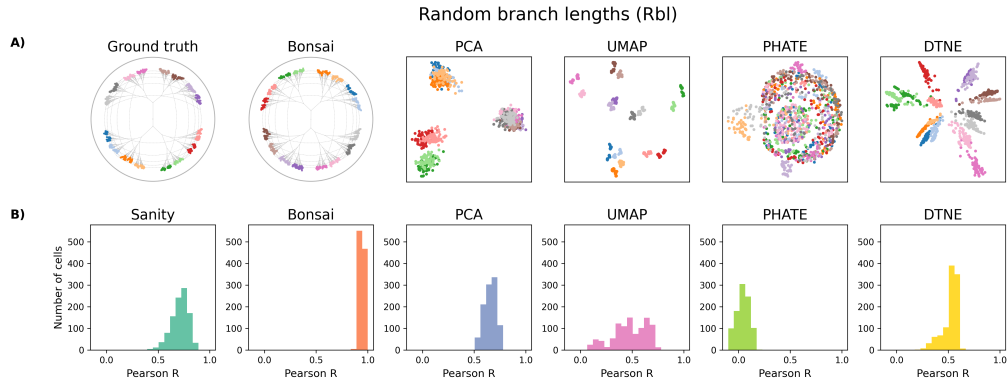


Figure S4. Visualization and pairwise distance recovery on a simulated dataset based on a binary tree with random branch lengths. **A)** Ground truth (left) and visualizations by different methods of simulated data of a tree with random branch lengths (see Section SI.E.2 for information on the simulation). **B)** Histograms of, for each method, the correlations (Pearson R) between the ground truth distances from each cell to all other cells and the corresponding distances in the visualization.

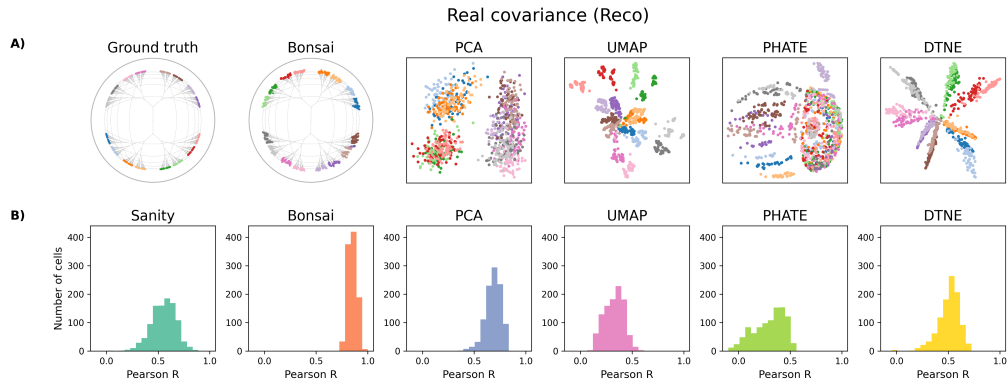


Figure S5. Visualization and pairwise distance recovery on a simulated dataset based on a binary tree using diffusion according to a covariance matrix estimated from a real dataset. **A)** Ground truth (left) and visualizations by different methods of simulated data of a tree with gene expression correlations matching the covariance matrix of a real dataset (see Section SI.E.2 for information on the simulation). **B)** Histograms of, for each method, the correlations (Pearson R) between the ground truth distances from each cell to all other cells and the corresponding distances in the visualization.

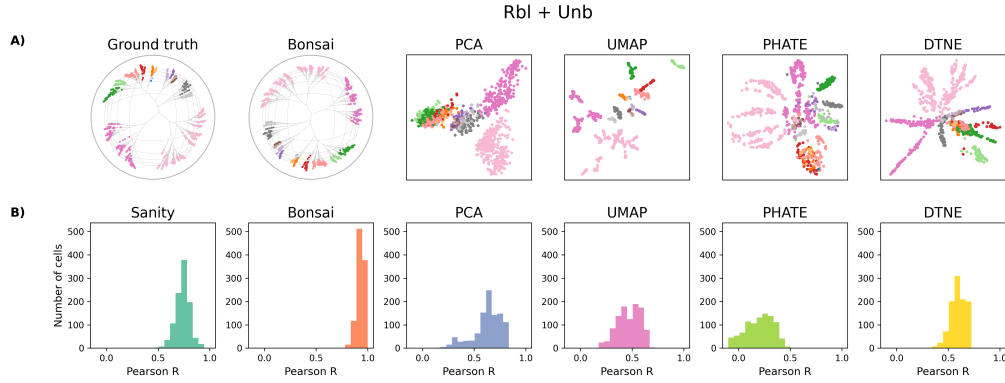


Figure S6. Visualization and pairwise distance recovery on a simulated dataset based on an unbalanced tree with random branch lengths. **A)** Ground truth (left) and visualizations by different methods of simulated data of an unbalanced tree with random branch lengths (see Section SI.E.2 for information on the simulation). **B)** Histograms of, for each method, the correlations (Pearson R) between the ground truth distances from each cell to all other cells and the corresponding distances in the visualization.

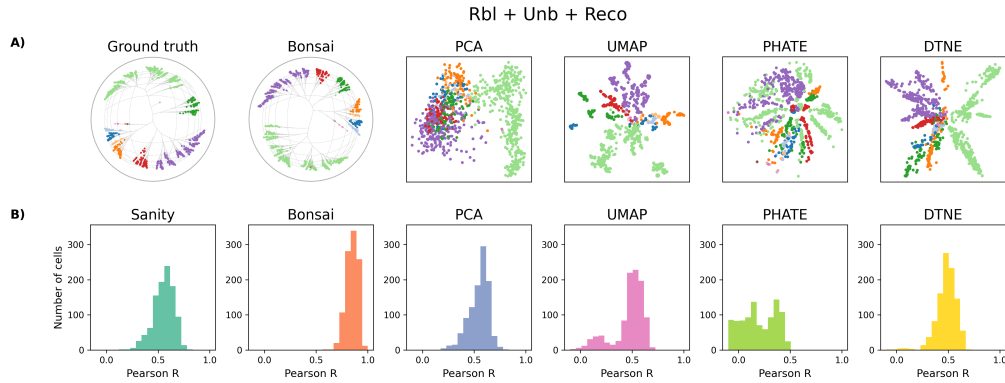


Figure S7. Visualization and pairwise distance recovery on simulated dataset based on an unbalanced tree with random branch lengths using diffusion according to a covariance matrix learned from a real dataset. **A)** Ground truth (left) and visualizations by different methods of simulated data of an unbalanced tree with random branch lengths and gene expression correlations matching the covariance matrix of a real dataset (see Section SI.E.2 for information on the simulation). **B)** Histograms of, for each method, the correlations (Pearson R) between the ground truth distances from each cell to all other cells and the corresponding distances in the visualization.

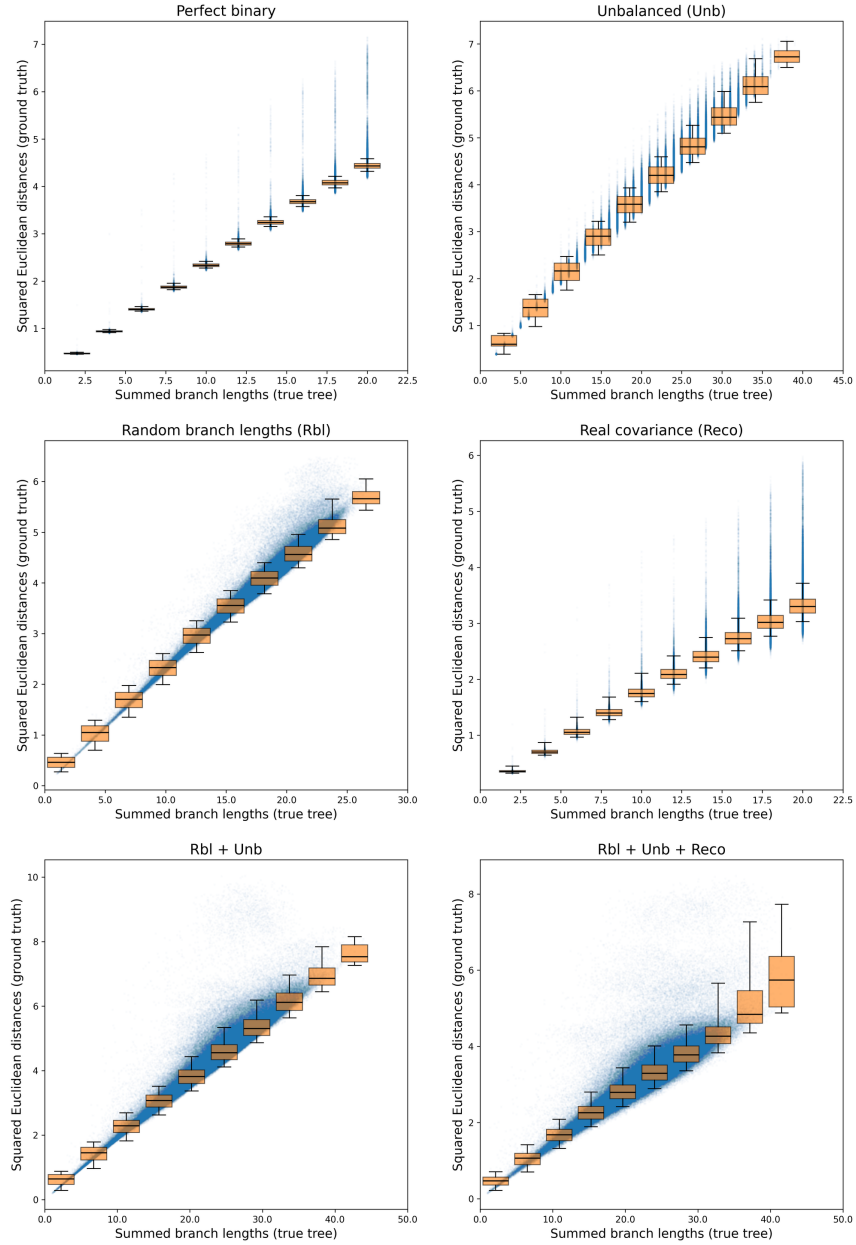


Figure S8. Distances along the ground truth tree accurately match the ground truth squared Euclidean distances. For the 6 simulated datasets described in Section S.I.E.2, we here show the linear relation between pairwise distances as measured along the ground truth tree (x-axis) and as measured by the squared Euclidean distance between the ground truth coordinates at the leaves (y-axis). Because there are 523 776 pairwise distances in these datasets, the scatter plots suffer from over-plotting. Therefore, we capture the linear relation by binning the data into 10 equal-sized bins along the x-axis, and then show a box-whisker-plot for each bin, indicating the 5, 25, 50, 75 and 95 percentiles.

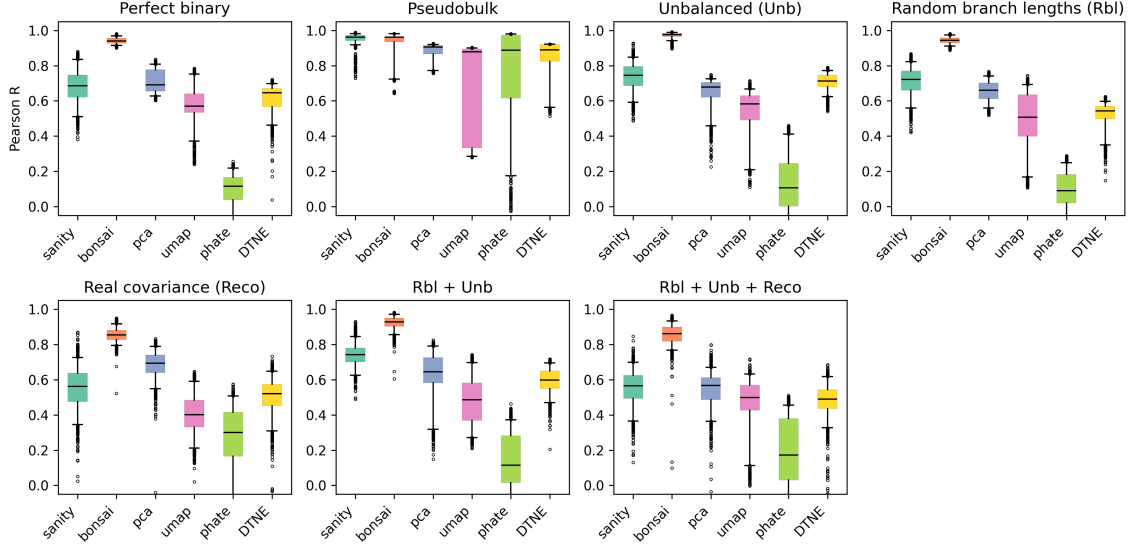


Figure S9. Performance of various visualization methods on preserving cell-to-cell distances. For the 7 simulated datasets described in S1E, and for which the detailed visualizations are shown in Figs S2-S7 and S11, we collect all calculated correlations between true and visualized cell-to-cell distances. Specifically, for each cell we compute the Pearson-R value for the correlation between the true and inferred distances between the cell and all other cells. The distributions of these Pearson-R values are captured in box-plots (with the box indicating the interquartile range and the whiskers the 5 and 95 percentiles across the $n = 1024$ cells). For PCA, UMAP, PHATE, and DTNE, the data are preprocessed with a standard log1p-normalization, after which a PCA-projection is performed. To present these methods in the best light, we performed a coarse parameter scan and for each method used the number of PCA components that performed best. This resulted in keeping all dimensions for UMAP, projecting to 500 PCs for DTNE, and 100 PCs for PHATE. The UMAP-embedding into two dimensions was run with parameters: `n_neighbors=15`, `min_dist=0.1`, `n_components=2`, `metric='euclidean'`, and we used the default parameters for DTNE and PHATE. For *Sanity* we used the specialized distance-calculation script, see [3].

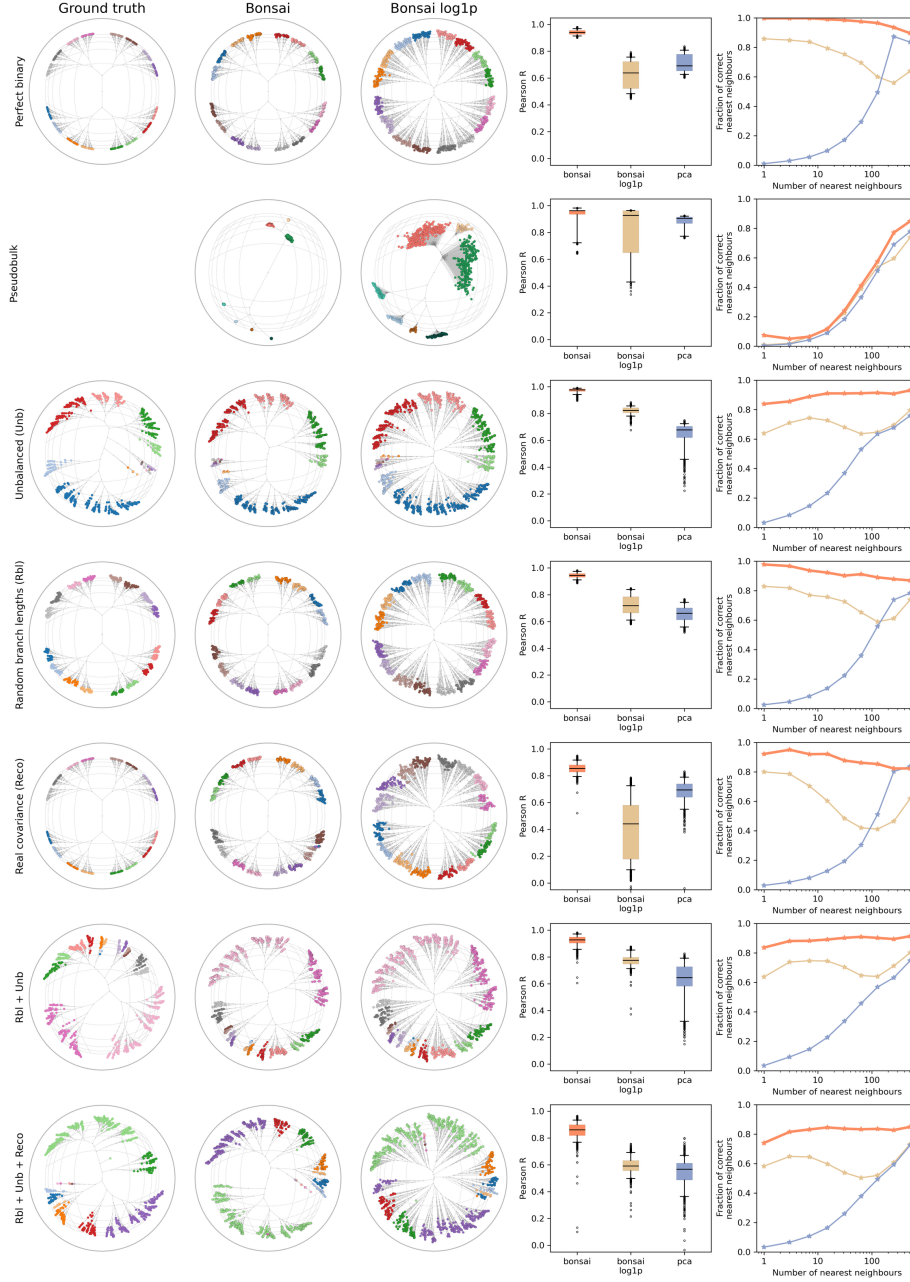


Figure S10. *Bonsai* reconstructions become less accurate if *Sanity*-preprocessing is replaced by conventional preprocessing, but remain more accurate than other visualization methods. Each row corresponds to one of the 7 simulated datasets described in SI.E. The columns show from left to right: the ground truth, the *Bonsai*-tree using default *Sanity*-preprocessing [3], the *Bonsai*-tree using a widely-used *log1p*-preprocessing pipeline (see Methods), box-whisker plots (indicating the 5, 25, 50, 75, and 95 percentile across $n = 1024$ cells) of the distributions of correlations between the true distances from each cell to all other cells and the inferred distances for the two versions of *Bonsai* and PCA for reference, and the performance on nearest-neighbor identification. We see that the *log1p*-based *Bonsai*-trees have terminal branch lengths that are notably longer. Because the *log1p*-preprocessing does not take into account the biological and technical noise, each cell appears distant from each other cell, even from those that have very similar ground truth states. Indeed, *Sanity*-based *Bonsai* strongly outcompetes *log1p*-based *Bonsai* on distance estimation and nearest-neighbor identification. It is remarkable, however, that even with this suboptimal preprocessing *Bonsai* is still much more accurate than the other visualization methods (only PCA is shown here, as it was the best-performing alternative to *Bonsai* according to Fig S9).

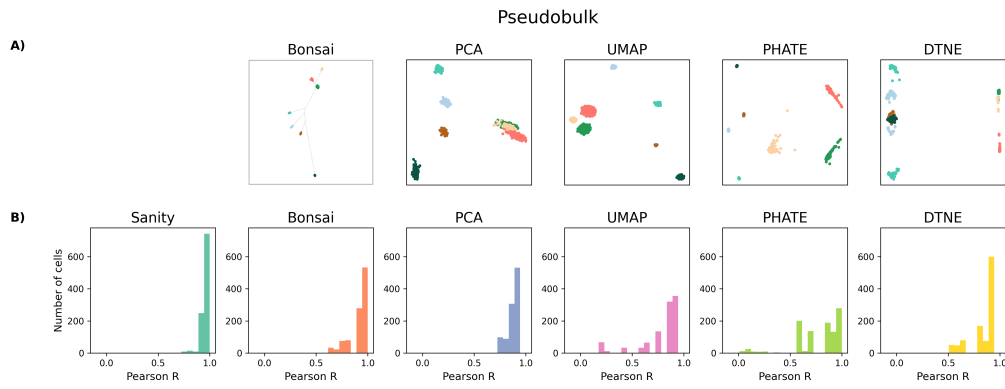


Figure S11. Visualization and pairwise distance recovery on a simulated dataset based on a pseudobulk dataset. **A)** Visualizations of a simulated dataset with 7 pseudobulk clusters of cells (colors) as obtained with different methods. Note that for this simulated dataset with 7 clusters of cells, there is no ground truth tree (see Section S1.E.2 for information on the simulation). **B)** Histograms of, for each method, the correlations (Pearson R) between the ground truth distances from each cell to all other cells and the corresponding distances in the visualization.

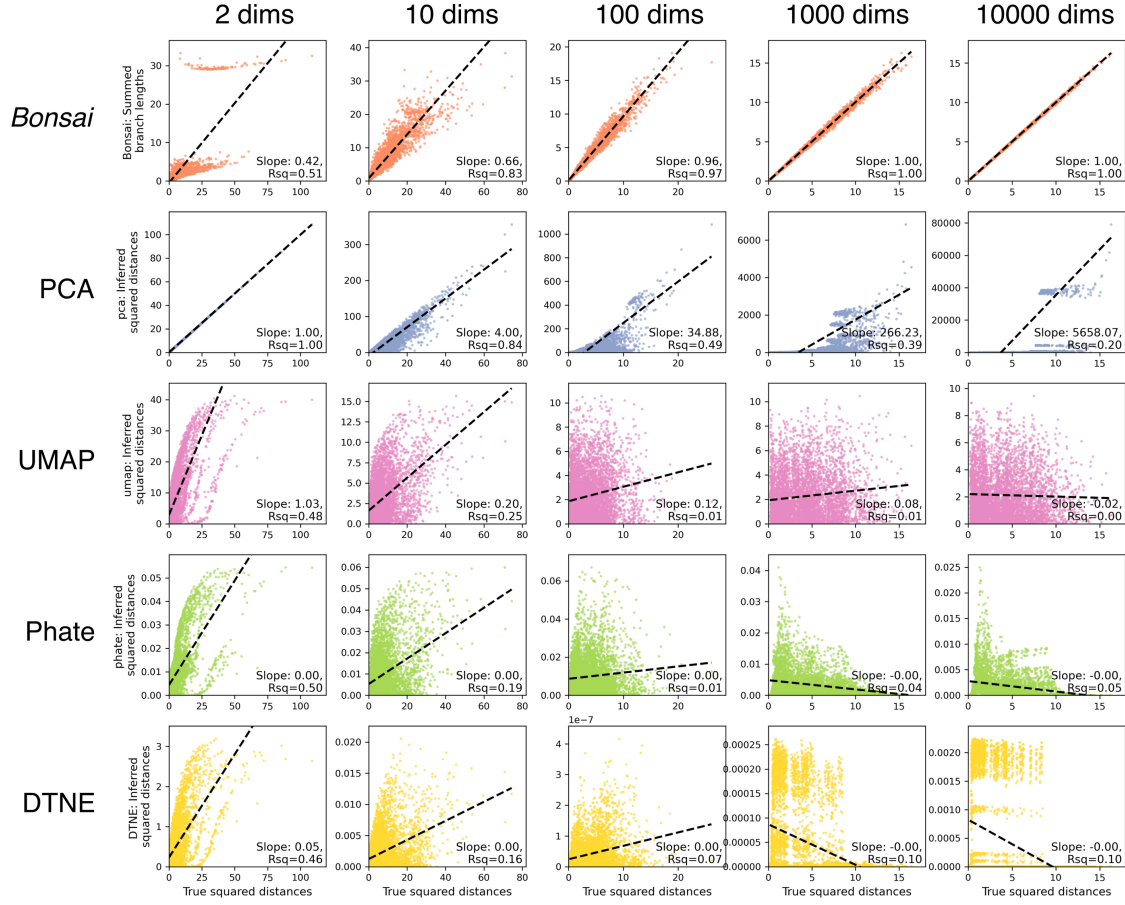


Figure S12. The blessing of dimensionality on random data: With increasing dimensionality, *Bonsai*-trees get better at representing distances while other visualization methods get worse. We created simulated datasets with random scatters of 100 cells in spaces of different dimensionality as follows. For each cell c we sample a variance v_c from a uniform distribution in log-scale between 0.1 and 10 ($\log \bar{v}_c \in [\log(0.1), \log(10)]$) and then normalize such that the average of the v_c equals one, i.e., $v_c = \frac{\bar{v}_c}{\frac{1}{|C|} \sum_{c'} \bar{v}_{c'}}$. For each gene g we sample a variance v_g from an exponential distribution with mean 2 and then sample log-expression values for each gene-cell pair (g, c) by drawing from a Gaussian with variance $v_{gc} = v_g v_c$. Each column of panels corresponds to a dataset with different dimensionality, increasing from 2 dimensions on the left to 10000 dimensions on the right. Rows correspond to different visualization methods with, from top to bottom: *Bonsai*, PCA, UMAP, PHATE, and DTNE. Each panel shows a scatter of the true pairwise distances between cells (horizontal axis) versus the inferred distances of the visualization method (vertical axis). Dotted lines show linear fits with slope and squared Pearson correlation indicated in each panel. As the dimensionality of the data increases, tree representations get more and more accurate, already reaching a very good correlation of $R^2 = 0.97$ in 100 dimensions. Instead, all other visualization methods become increasingly worse at representing distances in higher dimensions.

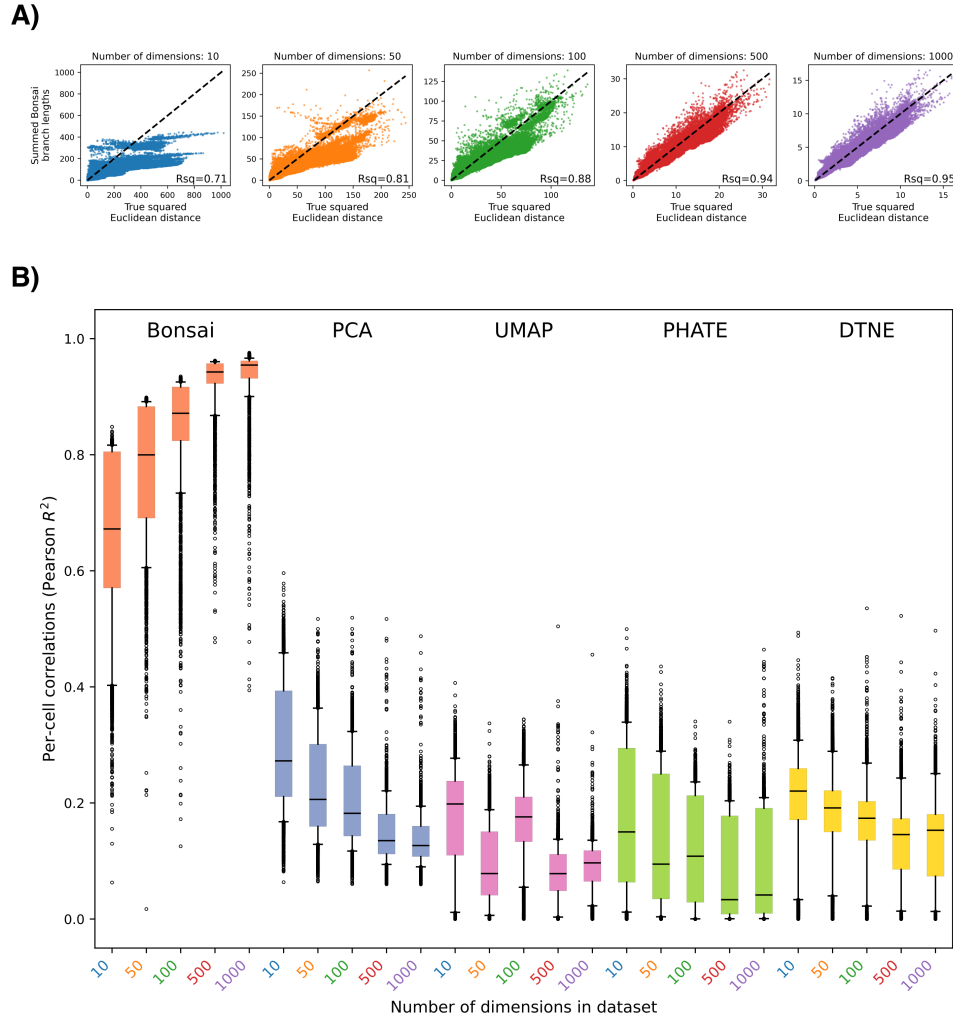


Figure S13. The blessing of dimensionality on real scRNA-seq data: Tree structures accurately capture true cell-to-cell distances in real scRNA-seq data, and improve as the dimensionality of the data increases. We performed analysis analogous to that of Fig. S12 using the gene expression values from a real scRNA-seq dataset of cord blood cells [4]. We processed the scRNA-seq data using *Sanity* [3] and treated *Sanity*'s estimates of the log-expression values as ground truth. To investigate the effect of the data's dimensionality, we created several datasets by projecting the data on the first 10, 50, 100, 500 and 1000 PCA components. We reconstructed a tree representation of each dataset using *Bonsai*, as well as 2-dimensional representations using PCA, UMAP, PHATE and DTNE. In **A)** we show scatters of the true squared Euclidean distances between pairs of cells, versus the squared-distances inferred by *Bonsai*, i.e., the sum of the branch lengths connecting the cells in the tree. The dashed line is the line $y = x$, and the Pearson R-squared value is reported in each plot. In **B)** we calculated for each cell the Pearson R^2 -value between the true and inferred distances to the other 999 cells. The box-whisker plots show the distributions of these R^2 -values (5, 25, 50, 75 and 95 percentile across 7509 cells). Note that the tree representations get more accurate as the dimensionality of the data increases while all other visualization methods deteriorate, even though the correlations for *Bonsai* are already much larger in the 10-dimensional dataset.

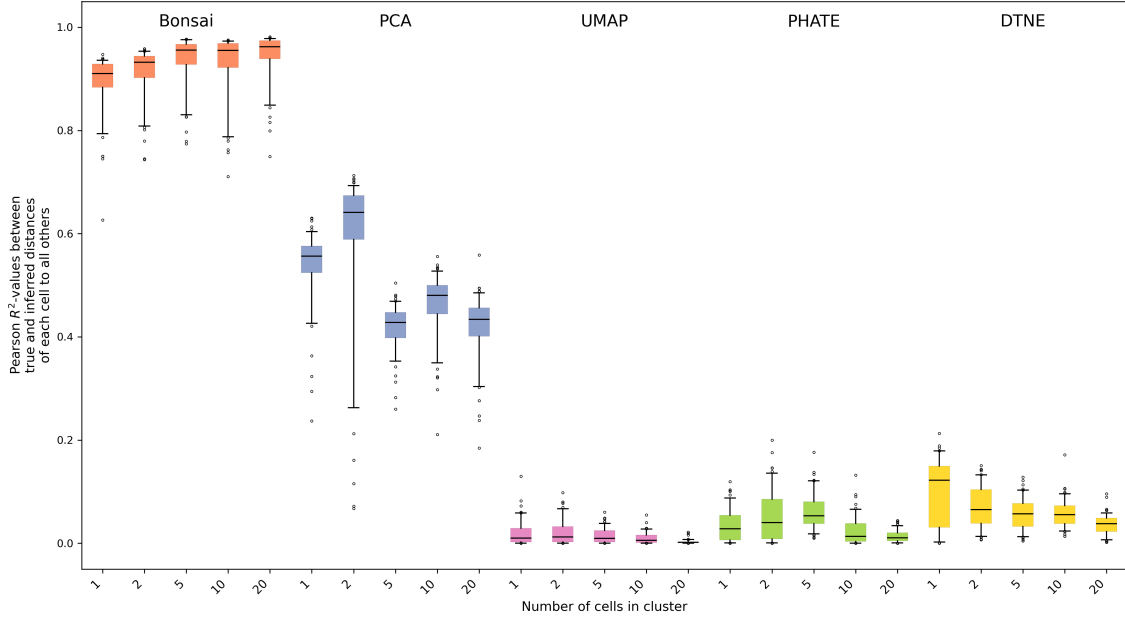


Figure S14. Tree-structures automatically use nearby cells to obtain more accurate distance estimates. We created synthetic datasets by first taking the 100 points in the 100-dimensional simulated dataset from Fig. S12 and using these as cluster-centers. We then sampled either 1, 2, 5, 10, or 20 cells around each cluster-center by adding a noise term drawn from a normal distribution in 100 dimensions with variance 2.5 in each dimension, see Section S.I.F.2 for details. We thus created datasets in which each cell has both a *measured position* (i.e., with the noise added) and a *true position* (i.e., equal to its corresponding cluster-center). We used *Bonsai*, PCA, UMAP, PHATE, and DTNE to construct a representation of the measured positions, and then checked how well these representations capture the pairwise distances between the *true positions*. In particular, to keep the test consistent across datasets, we selected only the first cell around each cluster-center and calculated the Pearson R^2 -value between true and inferred distances to the other 99 cells for each of the 100 cells. The box-whisker plots summarize the distribution of these squared correlation coefficients (5, 25, 50, 75, and 95 percentiles). We see that, apart from generally dramatically outperforming all other methods, *Bonsai*'s tree reconstruction approximates the distances from each cell to its neighbors better and better as the number of cells per cluster increases. This effect is completely absent or negative for the other methods. This further supports that the tree structure acts as a regularizer that improves the estimates of each cell's true position.

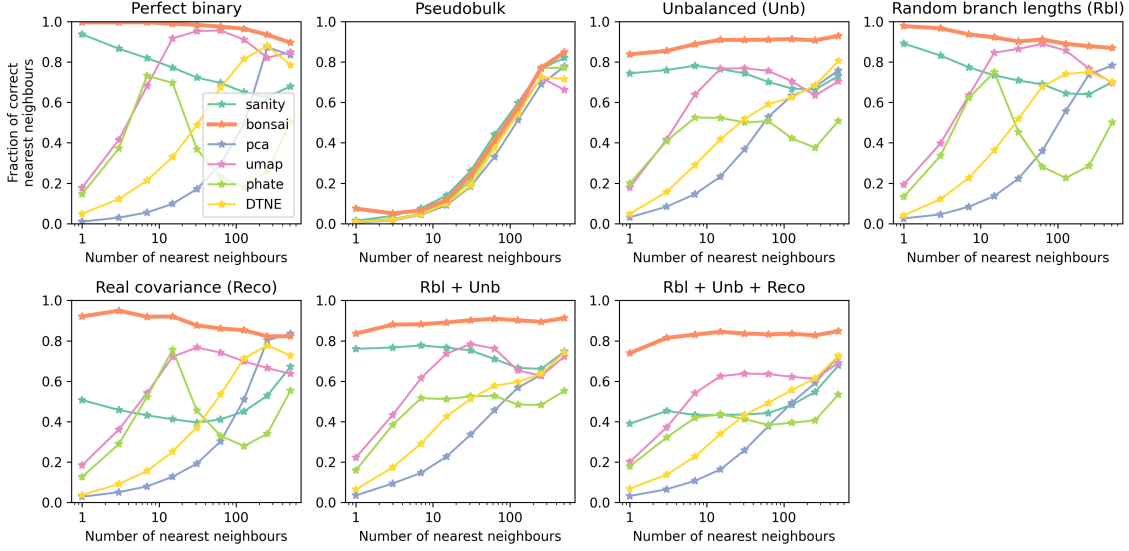


Figure S15. *Bonsai* outperforms other methods on identifying nearest-neighbor cells in scRNA-seq datasets. For the 7 simulated datasets described in S1.E, we show how well various methods identify the nearest neighbors of each cell. Specifically, for different numbers of nearest-neighbors k , we calculate what fraction of the inferred k nearest-neighbors are indeed among the k nearest-neighbors in the ground truth and show these fractions as a function of k . Each panel corresponds to one synthetic dataset (described at the top). For PCA, UMAP, PHATE, and DTNE, the data are preprocessed with a standard logp1-normalization after which a PCA-projection is performed to the number of components that, for each method individually, showed best performance in a parameter scan. This resulted in keeping all dimensions for UMAP, projecting to 500 PCs for DTNE, and 100 PCs for PHATE. The UMAP embedding into two dimensions was run with parameters: `n_neighbors=15`, `min_dist=0.1`, `n_components=2`, `metric='euclidean'`, and we used the default parameters for DTNE and PHATE. For *Sanity* we used the distance-calculation program that takes error-bars into account (see [3]). For the Pseudobulk dataset, all methods perform equally badly because no clear nearest-neighbor structure is present until the number of neighbors becomes larger than the typical cluster size. For all other datasets, *Bonsai* outperforms all other tools for virtually all values of k , often by a large margin. Notably, PCA and DTNE tend to perform well only for large k , suggesting that these methods only capture the large-scale structure in the data. UMAP and PHATE typically have a local peak in performance for a value of k somewhere between 30 and 100, reflecting that at these parameter settings these methods conserve cell-to-cell-structures at one particular scale only. Notably, *Bonsai* also outperforms *Sanity* on all datasets, even though both methods are based on the same inferred gene expression values. This suggests that *Bonsai*'s increased performance results from implicitly using the tree structure as a prior for estimating distances.

Reverse cumulative distribution of scRNA-seq dataset sizes.

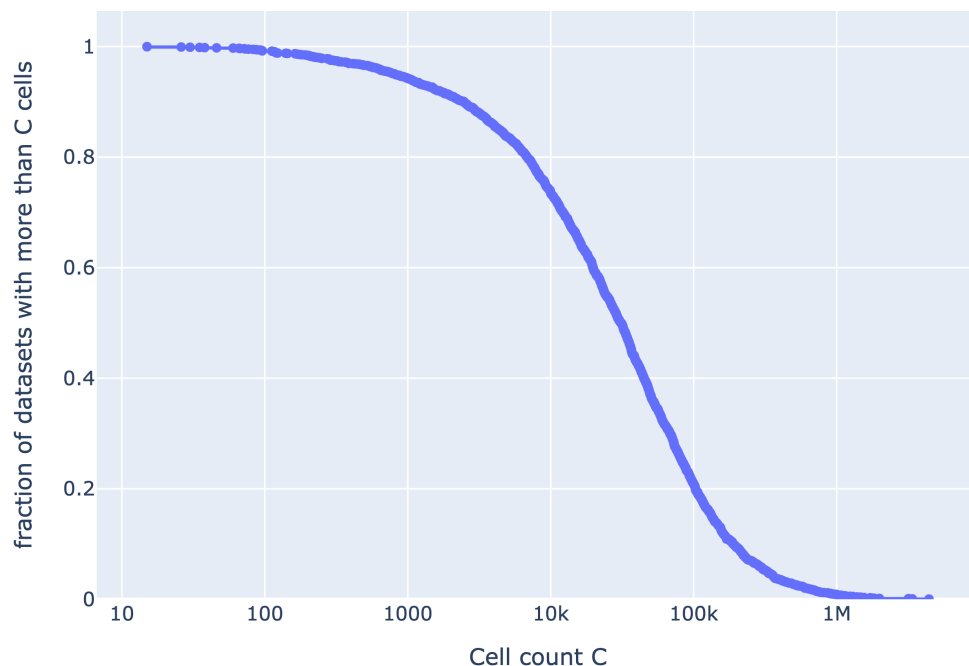


Figure S16. The distribution of cell numbers of scRNA-seq datasets across four public databases. We took all the scRNA-seq datasets from the databases Single Cell Portal (624 datasets) [5], DISCO (1323 datasets) [6], Single Cell Expression Atlas (380 datasets) [7], and PanglaoDB (236 datasets) [8], and here plot the reverse cumulative distribution of the cell numbers across all datasets, with the horizontal axis on a logarithmic scale. This shows that only a small minority of the datasets contain over 100 000 cells, and datasets with more than a million cells are very rare.

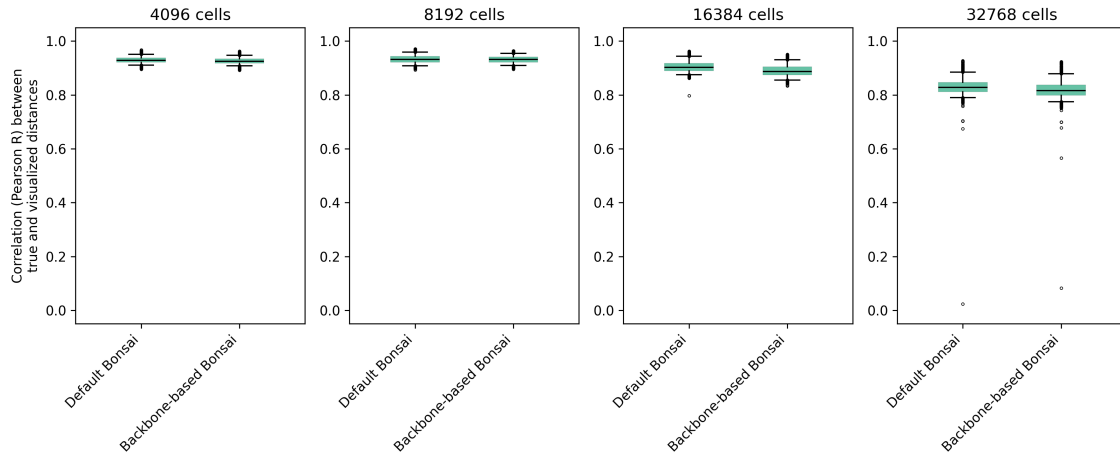


Figure S17. Trees reconstructed with Backbone-Based-Bonsai preserve cell-to-cell distances almost as well as standard *Bonsai*. In the time- and memory-scaling tests described in Fig. 3, there were four simulated datasets on which we ran both the default *Bonsai*-algorithm and Backbone-Based-Bonsai. To show that the computationally efficient backbone-based algorithm is still accurate, we here compare the visualized distances (sum of branch lengths) with the ground truth cell-to-cell distances. Specifically, the box-whisker plots capture the distribution over cells of the correlations between the true and inferred distances between the cell and all other cells (5, 25, 50, 75, and 95 percentiles). We see that, in all four datasets, the visualized distances are very accurate regardless of which algorithm was used to reconstruct the tree.

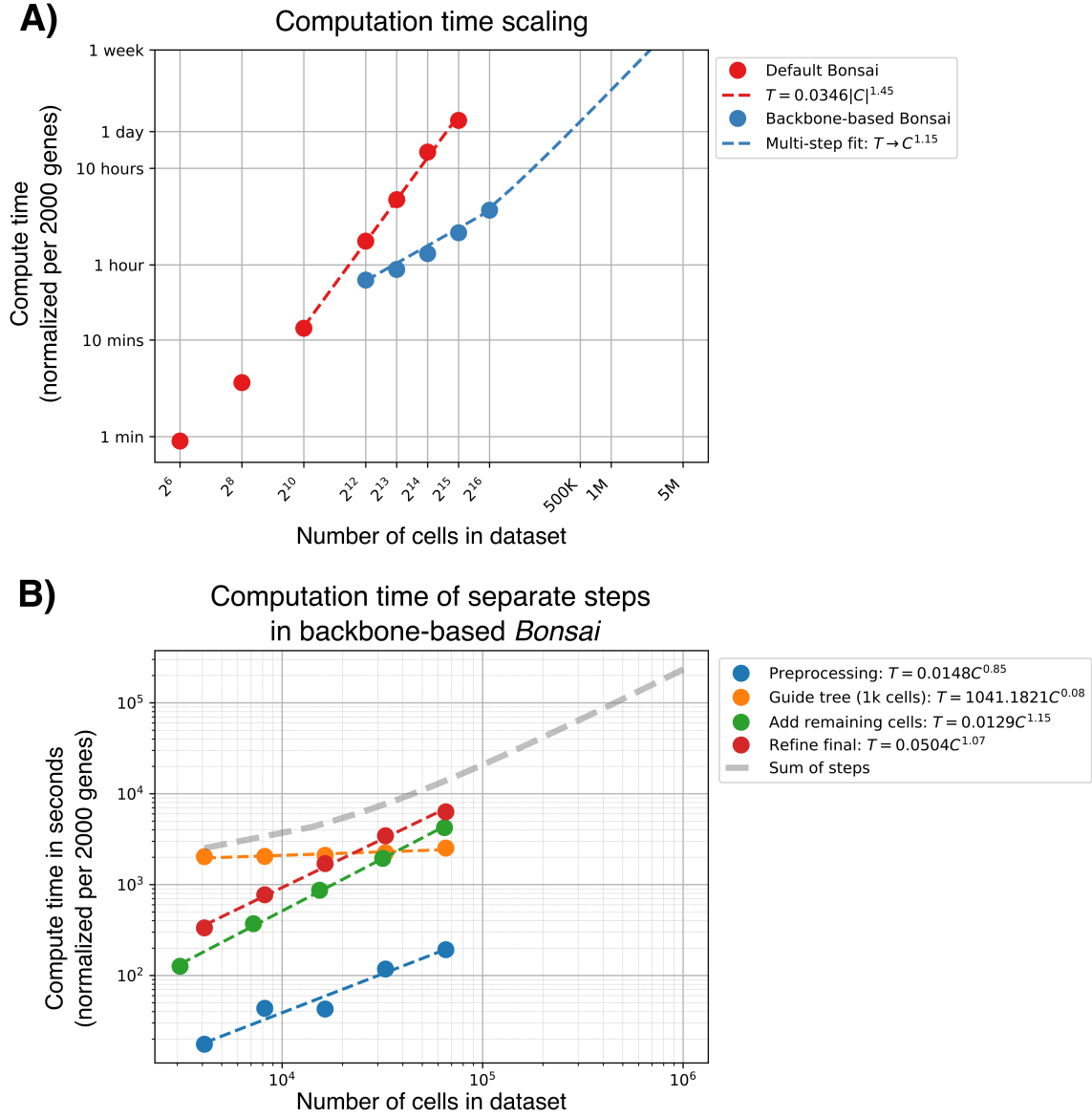


Figure S18. Scaling of computation-time for the default *Bonsai* algorithm, and the Backbone-Based-*Bonsai* algorithm. **A)** The computation times for datasets of different sizes simulated following the procedure described in S1.E.2.1. Since *Bonsai* starts by selecting only genes with a signal-to-noise ratio of 1, datasets differ slightly in their number of genes, although this number is close to 2000 in all datasets. To get comparable scaling, and because computation time scales linearly in the number of genes, we therefore correct the actual computation times as if 2000 genes were selected. While the default *Bonsai* algorithm shows a simple scaling behavior: $T \propto C^{1.45}$, this is more complicated for Backbone-Based-*Bonsai* because it consists of several steps with different computational requirements. We therefore fit the computational scaling of these individual steps in **B)**, so that their sum gives the overall scaling behavior. The 4 steps of backbone-based *Bonsai* are: 1) preprocessing the data (blue), 2) using default *Bonsai* to build a backbone-tree on 1000 cells (orange), 3) placing the remaining cells on the backbone (green), and 4) refining the full tree (red). The fits show that the scaling will eventually be dominated by the step in which remaining cells are placed on the backbone.

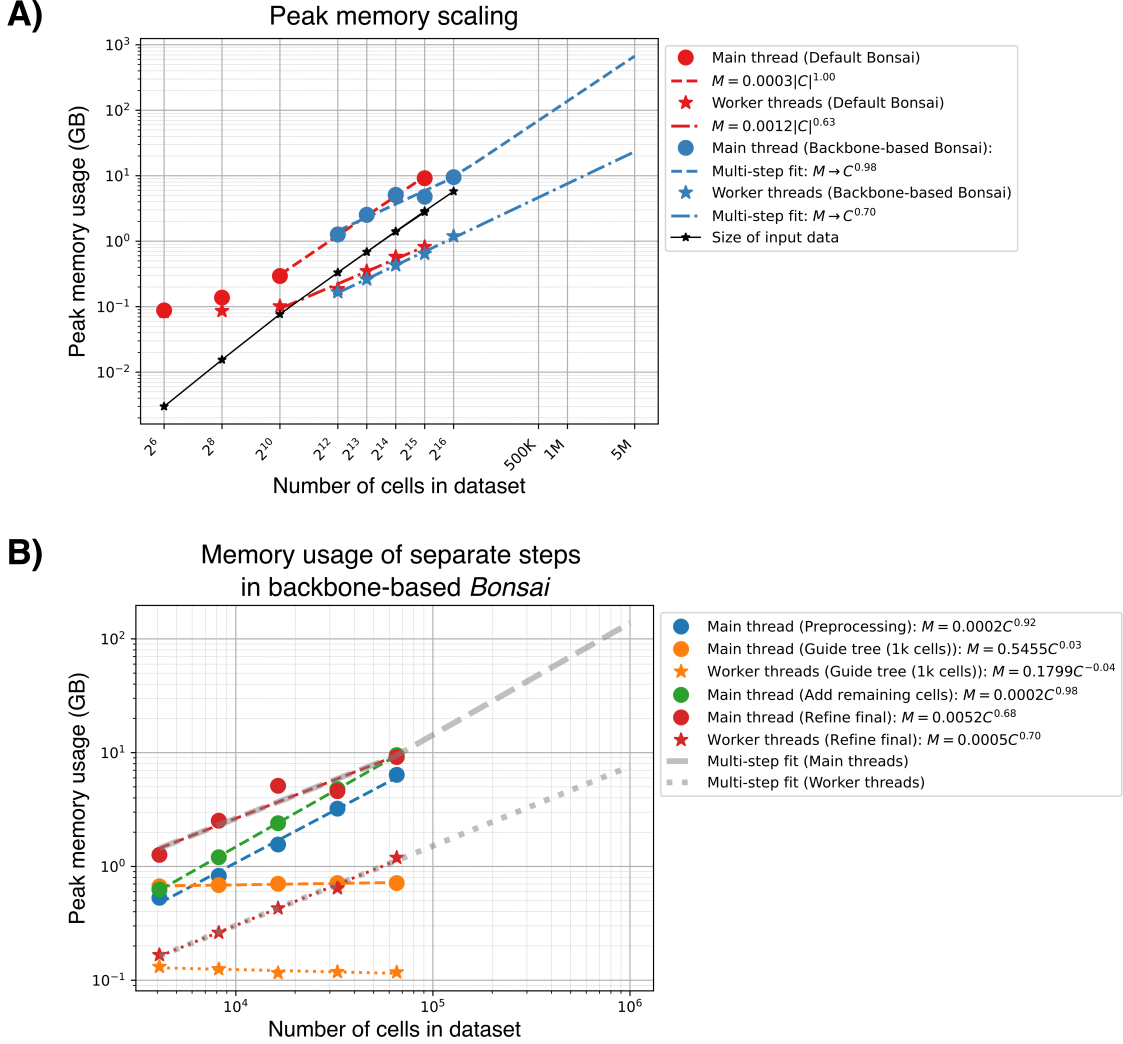


Figure S19. Scaling of peak memory usage for the default *Bonsai* algorithm, and the Backbone-Based-*Bonsai* algorithm. **A)** The peak memory usage of *Bonsai* while working on datasets of different sizes simulated following the procedure described in SI.E.2.1. Note that these peak memory usages are reached only at specific steps in the computation, while average memory usage is much lower. Both *Bonsai* algorithms are not parallelized symmetrically over different CPUs. Instead, there is a main thread that does most of the simple computations and also requires access to the data (non-sparse feature and uncertainty matrices) for all cells. The worker threads are only used for large calculations, and then only need access to the data of a subset of cells. This keeps the total memory requirement of all CPUs combined lower. Therefore, we separately report the peak memory usage of the main thread and the average of the worker threads. **B)** Similar to the computation time analysis in Figure S18, we fit the memory scaling of the different steps of Backbone-Based-*Bonsai* separately, but now the multi-step fit is not given by the sum of the steps, but by the maximum.

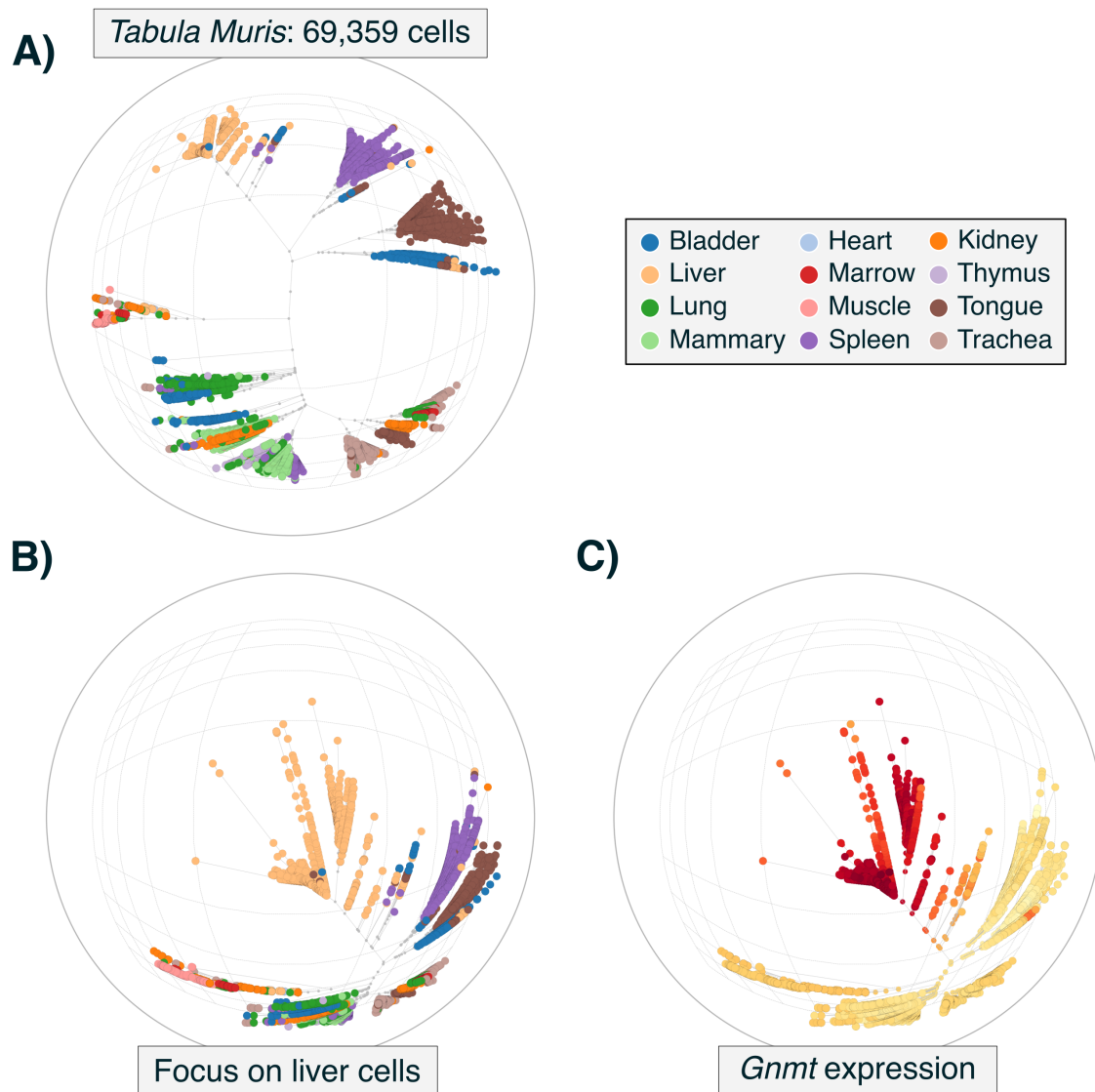


Figure S20. Bonsai reconstruction of 69 359 cells from the Tabula Muris dataset [9]. A) We show the *Bonsai* tree reconstructed for the *Tabula Muris* dataset in a circular layout on the hyperbolic disk, with different colors indicating the different tissues that the cells originate from. We have made these *Bonsai* results available for exploration using *Bonsai-scout* at https://bonsai.unibas.ch/bonsai-scout/?dir=Tabula_Muris as a resource for the community. The *Bonsai-scout* dashboard can for example be used to zoom in on the clade of liver cells **B)**, or to detect and visualize marker gene expression **C)**.

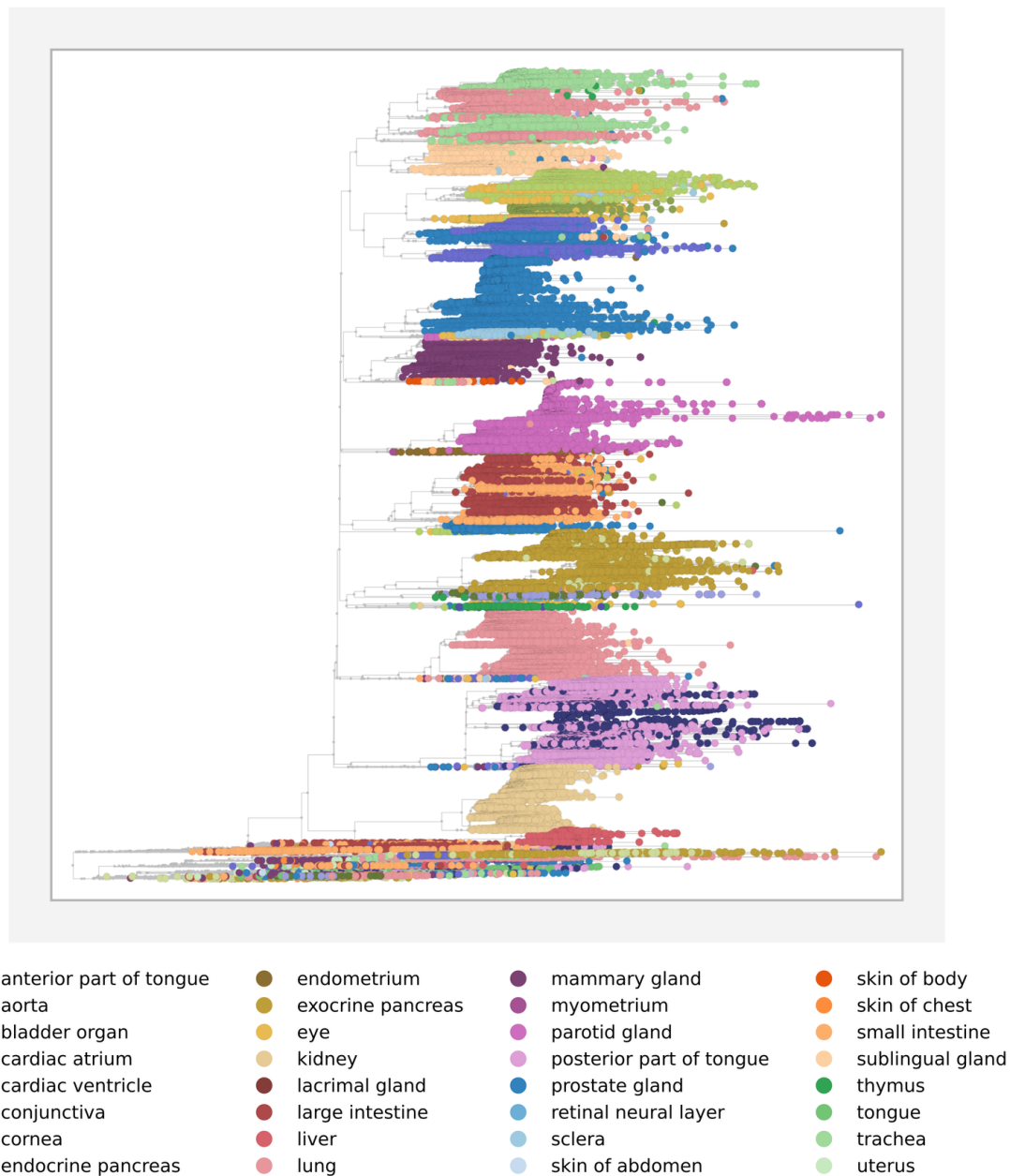


Figure S21. Bonsai reconstruction of 104 148 epithelial cells from Tabula Sapiens [10]. A dendrogram layout of the *Bonsai* tree reconstructed on the subset of all epithelial cells from the *Tabula Sapiens* atlas, with different colors indicating the different tissues that the cells originate from. We have made these *Bonsai* results available for exploration using *Bonsai-scout* at https://bonsai.unibas.ch/bonsai-scout/?dir=Epithelial_cells_from_Tabula_Sapiens as a resource for the community.

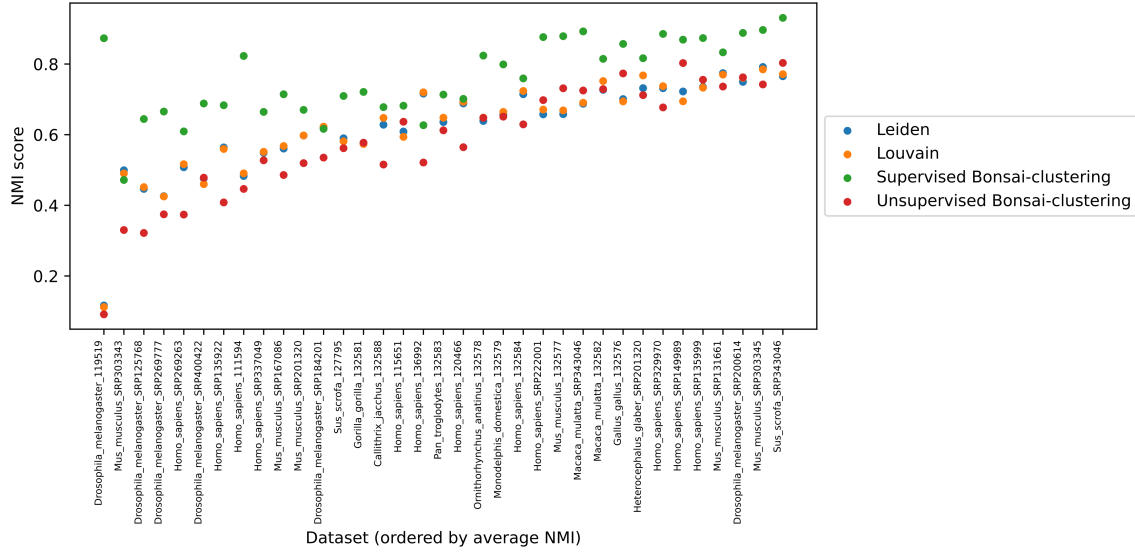


Figure S22. Performance in recovering reference annotations for *Bonsai-scout*'s supervised and unsupervised clustering methods compared with conventional Leiden and Louvain clustering. We ran *Bonsai* on all scRNA-seq datasets with fewer than 100 thousand cells from the BGee database [11], which all come with an author-provided cell type annotation. We then ran both the supervised and unsupervised clustering methods implemented in *Bonsai-scout*, as well as two standard clustering methods: Louvain and Leiden. To quantify how well these clustering methods matched the author-provided annotation, we here show the Normalized Mutual Information between the inferred clusters and the author-provided annotation for each dataset and each method (see legend). Datasets were sorted from left to right by average NMI across the clustering methods. The annotations for the datasets on the right side of the plot thus generally agree better with all of the clustering methods, which may indicate that these annotations are more trustworthy. *Bonsai*'s supervised method generally outperforms the other methods, while, on the left side of the plot, *Bonsai*'s unsupervised clustering performs slightly worse than Leiden and Louvain clustering. However, the user-provided annotations may have often been obtained with these clustering methods.

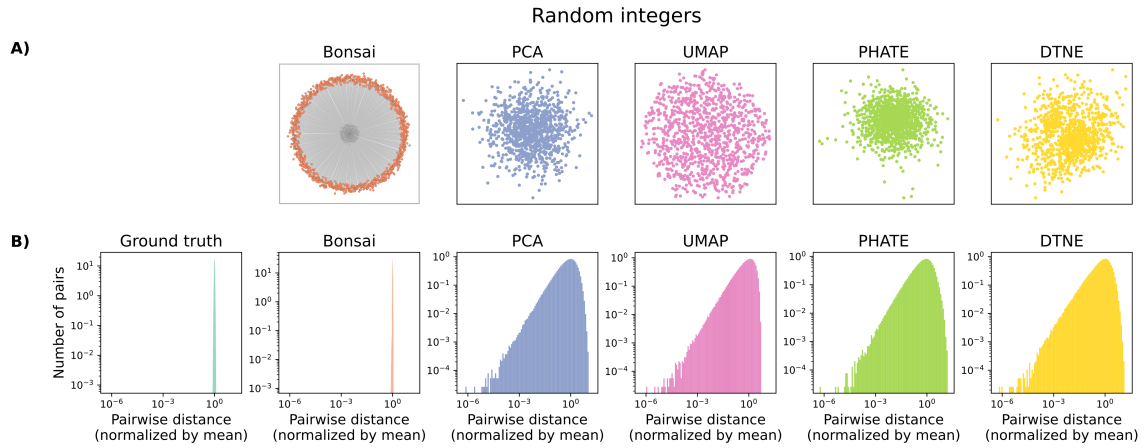


Figure S23. *Bonsai* accurately detects the absence of structure in a dataset. We simulated a dataset by randomly picking a count between 0 and 1000 for 512 genes and 1024 cells. **A)** Visualizations of this dataset created by *Bonsai*, PCA, UMAP, PHATE, and DTNE. In such random high-dimensional data, cells are expected to all be equidistant, which is captured by the *Bonsai* tree which connects all cells through long branches with a single root. The other visualization methods, however, fail at this task. **B)** Distributions of pairwise cell-to-cell distances for the different visualizations, where we normalized by the mean to make the histograms comparable. We indeed confirm that the spread in pairwise distances is almost non-existent in the ground truth and in *Bonsai*, while it is quite substantial in the other visualization methods. As a result, these methods hallucinate structure that is not present in the high-dimensional data.

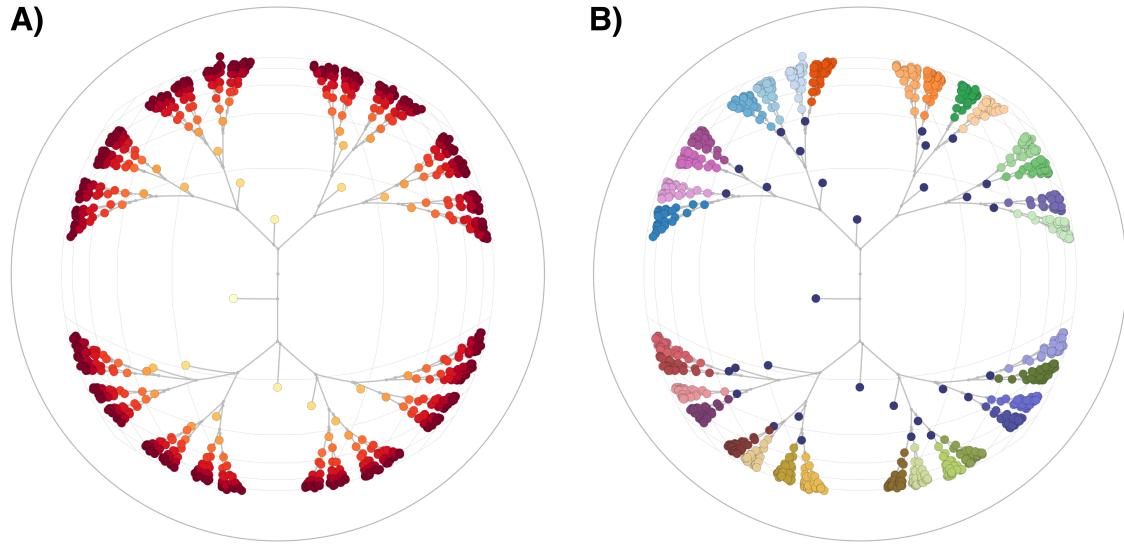


Figure S24. *Bonsai* correct places cells in precursor states near the corresponding internal nodes. Above we simulated a differentiation process by modeling gene expression dynamics along a binary tree (see Section SI.E for details). While this earlier simulated dataset only contained cells from the tips of the tree, we now create a dataset where cells are added for each of the internal nodes of the tree. **A)** *Bonsai* reconstruction with cells colored by their ground truth ‘depth’, i.e., the number of generations they were from the root in the ground truth. Yellow cells are thus early progenitors, while dark red cells are terminally differentiated cells. We see that the cells in precursor states are placed close to the internal nodes that they corresponds to. **B)** the same *Bonsai* tree with cells colored by which of the 32 terminal clades they belong to in the ground truth tree (i.e., defined by the first 5 branching points). Dark blue cells are upstream of at least 2 of these clades and thus have no clade membership. The perfect accordance of colors in the 32 clades shows that the precursor cells are placed next to the *correct* internal nodes.

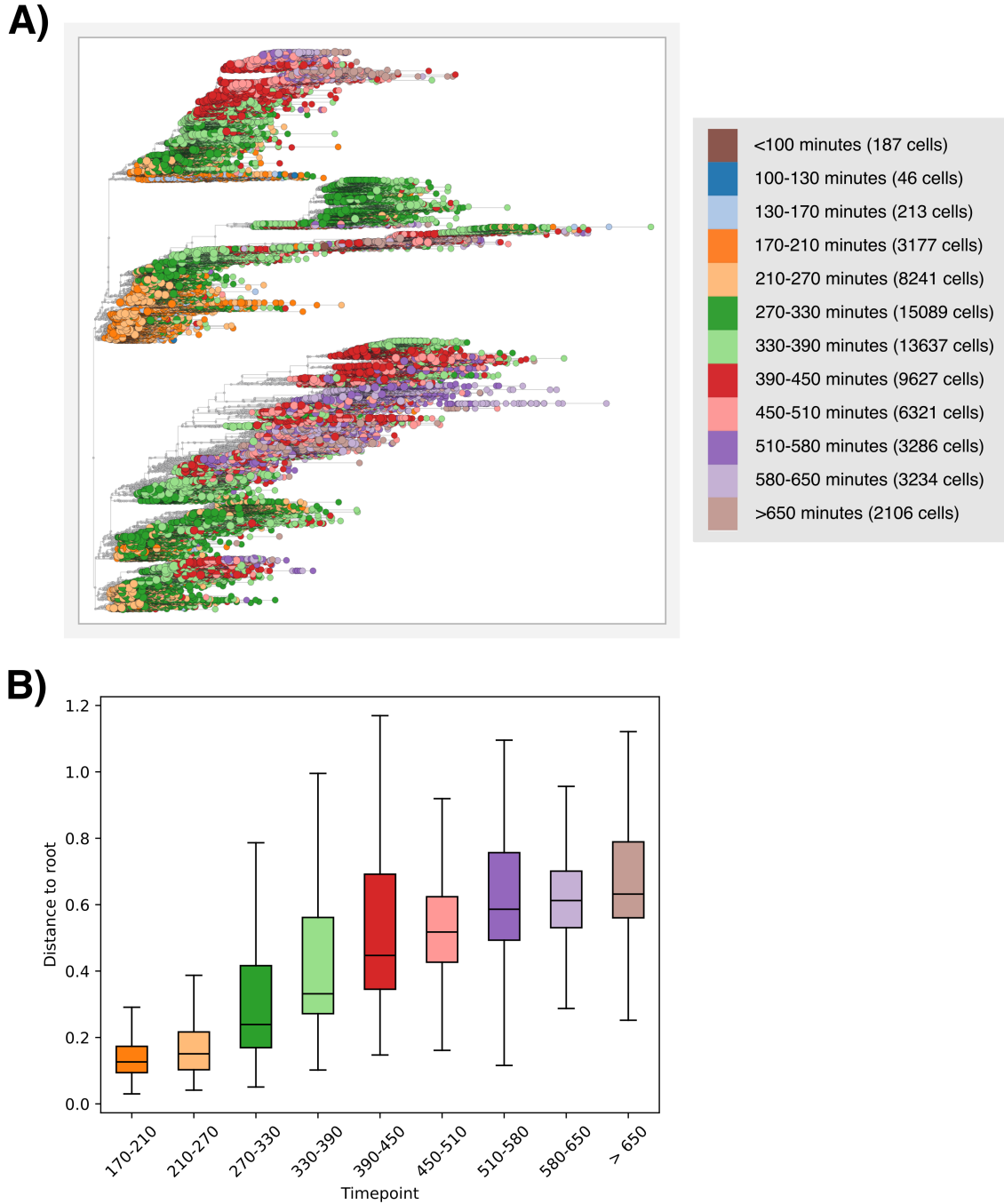


Figure S25. *Bonsai* places *C. elegans* cells from earlier developmental stages upstream of cells from later developmental stages. We reconstructed a *Bonsai*-tree on the *C. elegans* embryogenesis dataset from Packer et al. [12], restricting ourselves to the 65 164 cells from the Waterston lab (out of 89 701 cells total). This dataset contains cells from multiple embryos at different developmental stages and the authors provide an estimated developmental stage for each cell (based on comparison with a bulk RNA-seq time series). **A)** *Bonsai*-tree with cells colored by their annotated developmental stage. The tree shows several well-separated clades corresponding to lineages that give rise to different cell types. The annotation shows that the developmental stages of the cells tend to increase with distance from the root, i.e. from left to right. **B)** We quantify this by showing the distribution of distances to the root of the *Bonsai* tree for cells at each developmental stage, confirming that the location in the *Bonsai*-tree reflects developmental stage. Box-whisker plots correspond to 5, 25, 50, 75, and 95 percentiles. Cell numbers in each class are in the legend of panel A.

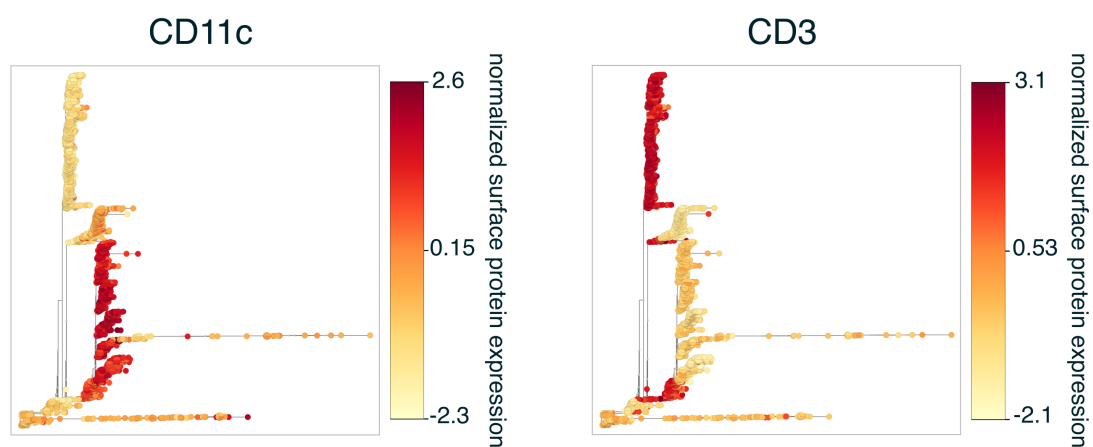


Figure S26. Surface marker abundance confirm the myeloid- and T-cell identity of clades in the cord blood *Bonsai* tree. Because the *Bonsai*-tree shown in Figure 4 was reconstructed solely based on scRNA-seq counts, we can use the second modality in this dataset as independent validation, i.e., the protein surface marker counts. We here plot the abundance of myeloid marker CD11c (left panel), and of the T-cell marker CD3 (right panel) on the *Bonsai* tree.

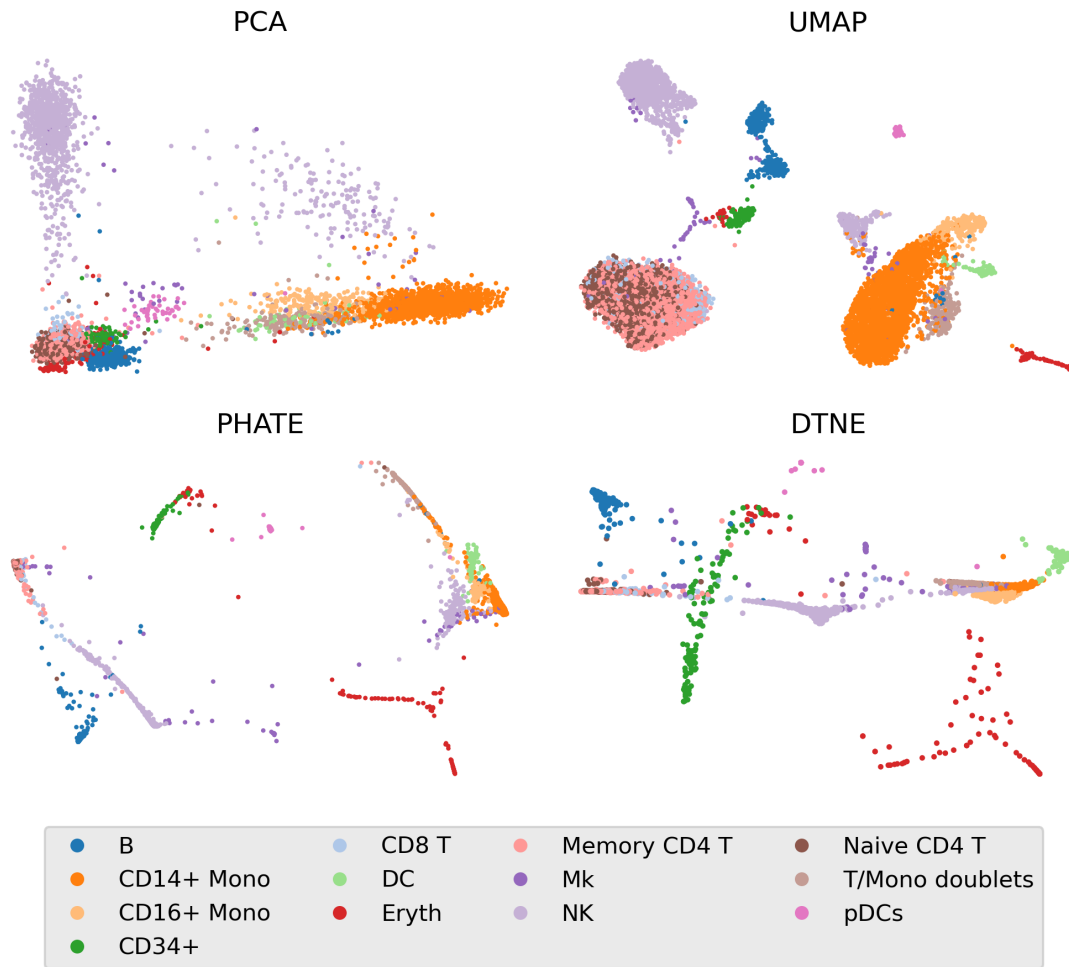


Figure S27. Other visualization methods do not recover differentiation trajectories of blood cell types. We used the visualization methods PCA, UMAP, PHATE, and DTNE, to create two-dimensional embeddings for the cord blood dataset. The cells are colored by the RNA-based cell type annotations provided with the original publication. As is typically observed, cells annotated as the same cell type are often placed close together in ‘blobs’ or ‘streaks’ in these visualizations, although with large variation in their spread and large disagreements in both the relative placements and spread of different cell types across the methods, e.g. the shape and position of the CD34+ hematopoietic stem cells. Crucially, very little of the known differentiation trajectories can be unambiguously identified in these visualizations, e.g. none of the visualizations identifies that erythrocytes split off first from the CD34+ HSCs. At best, the visualizations more or less indicate the major division between the lymphoid and myeloid lineage cells. PCA, UMAP, and PHATE were computed with *Scanpy* using standard preprocessing. DTNE was computed by applying a log1p transformation to raw counts, performing PCA, and then running DTNE on the top 100 principal components.

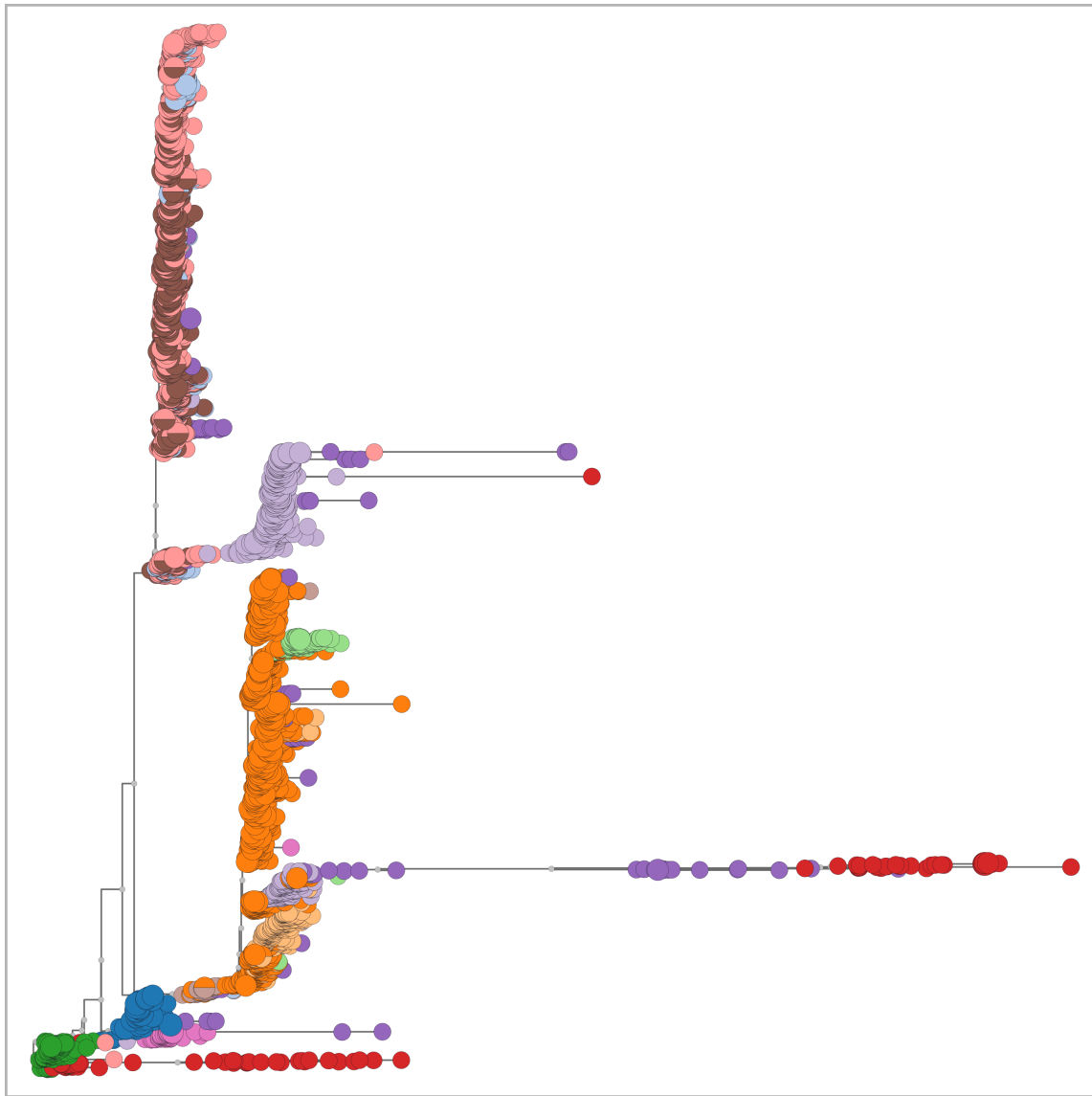


Figure S28. Removing potential doublet cells does not cause major changes to the predicted differentiation trajectories. We removed all cells flagged as potential doublets by *Scrublet* [13] and reran *Bonsai* on this reduced dataset. In total, 401 (of 7509) cells were excluded, including 151 that were annotated as T/Mono doublets in the original publication. In contrast, from the much larger group of NK-cells, only 77 were removed. The tree structure remains relatively unchanged. In particular, the two NK subgroups remain clearly visible after doublet removal, confirming that these two subtypes of NK cells are not an artefact caused by doublets.

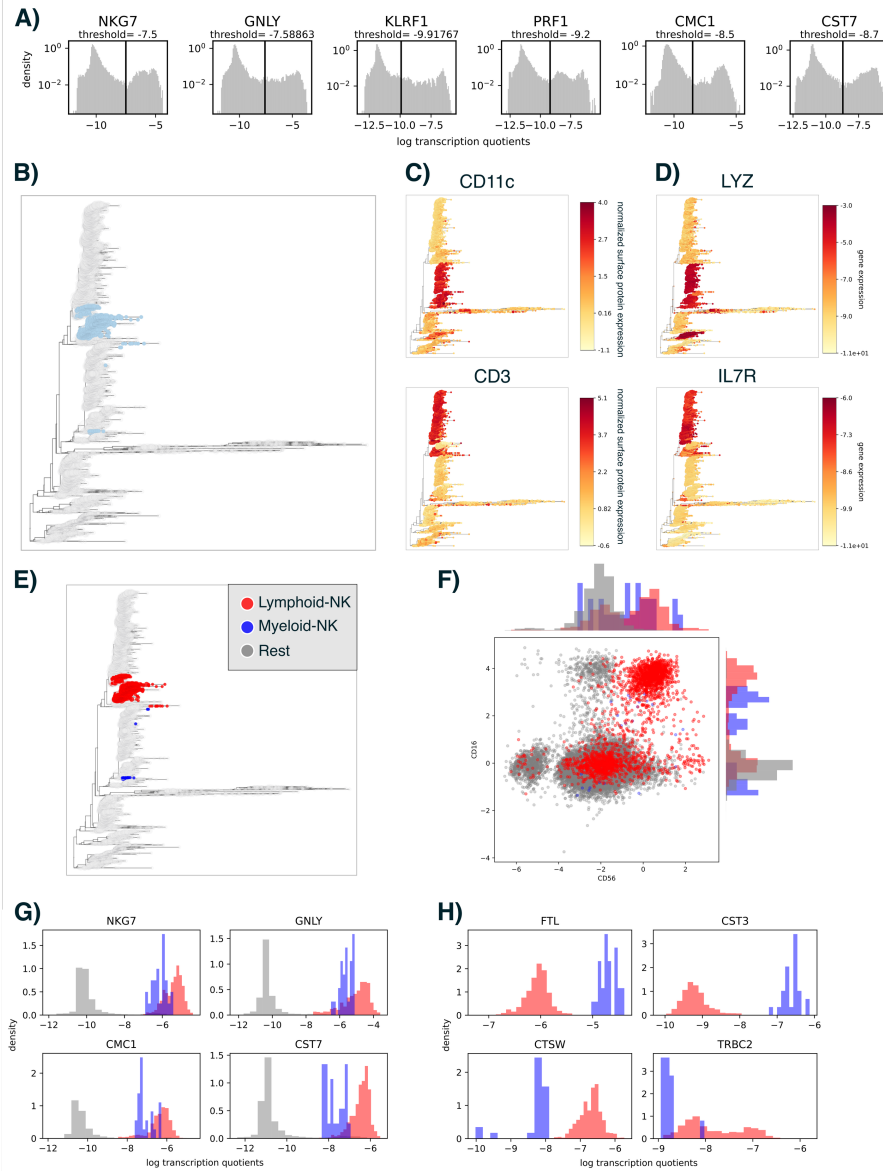


Figure S29. A small fraction of bone marrow-NK cells shows strong characteristics of deriving from the myeloid lineage. **A)** Histograms of gene expression (log transcription quotients) for 6 canonical NK marker genes across all 33 403 cells in the bone marrow dataset of Stuart et al. [14]. The black vertical lines indicate thresholds used to classify NK cells: cells are defined as NK if their gene expression is over the threshold for all six markers (2944 cells). **B)** Bonsai-tree colored by the NK cell classification defined in panel A. **C-D)** Overlaying the surface marker abundance of CD11c (myeloid marker) and CD3 (T cell marker), and the mRNA expression of LYZ (myeloid marker) and IL7R (lymphoid marker) genes identifies the myeloid and lymphoid branches. **E)** NK cells are found in two clades with one within the lymphoid branch (red, 2467 cells) and one clade within the myeloid branch (blue, 27 cells). **F)** A scatter plot of ADT expression for NK surface proteins CD56 and CD16 confirms the increased expression of these surface markers in the NK-annotated cells, although both subpopulations contain cells where these markers are absent. **G)** Four known NK marker genes are indeed upregulated in both the lymphoid-NK and myeloid-NK subpopulations. Interestingly, these markers exhibit slightly lower expression in the myeloid-NK population in both this dataset and the cord blood dataset (Fig. 4). **H)** Four of the marker genes that were detected to distinguish myeloid-NK and lymphoid-NK cells in the cord blood dataset also distinguish these populations in this bone marrow dataset.

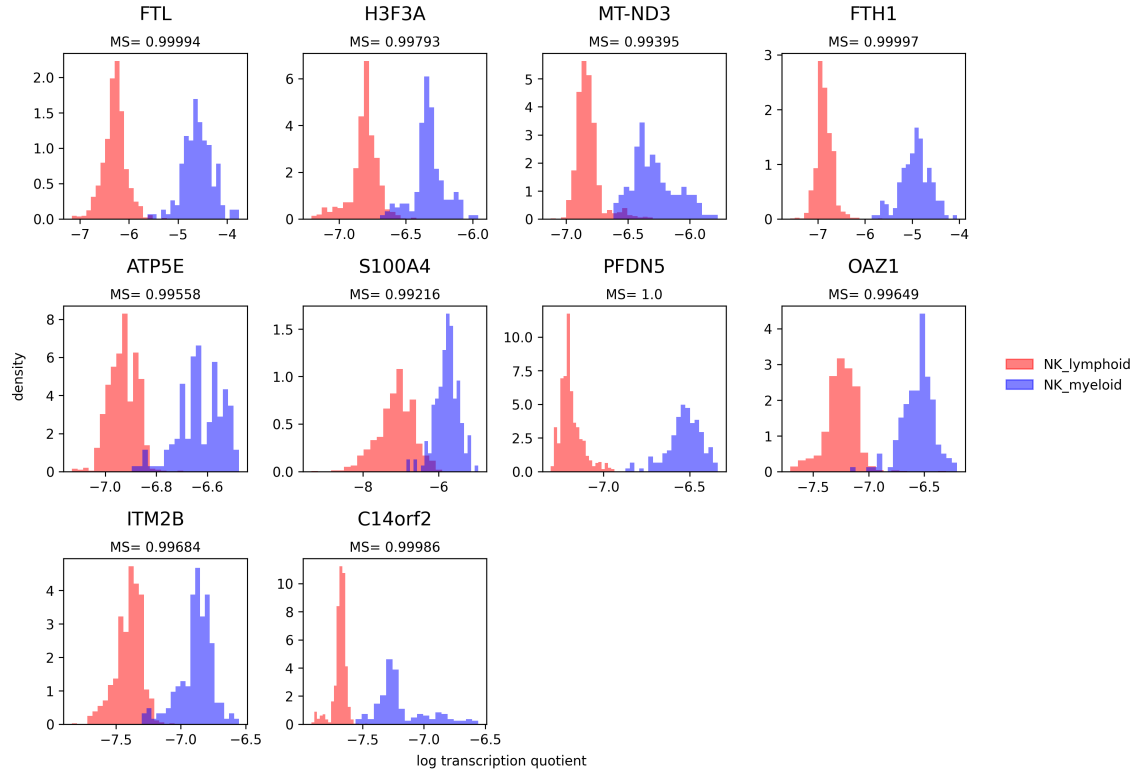


Figure S30. Marker genes that are consistently higher expressed in myeloid-NK cells than lymphoid-NK cells. For the two subpopulations of NK cells identified by the *Bonsai* representation of the blood cell dataset [4] (see Fig. 4), we found all marker genes with a marker score above 0.99 using our specialized marker-gene-detection script. The marker score (MS) is the probability that the gene is higher expressed in a randomly chosen myeloid-NK cell than in a randomly chosen lymphoid-NK cell (see Methods). We here present the 10 marker genes that had the highest mean expression in the myeloid-NK cells. Each panel corresponds to one marker gene, and the red and blue histograms show the distribution of expression levels (log transcription quotients) in the 1925 lymphoid-NK and 155 myeloid-NK cells, respectively.

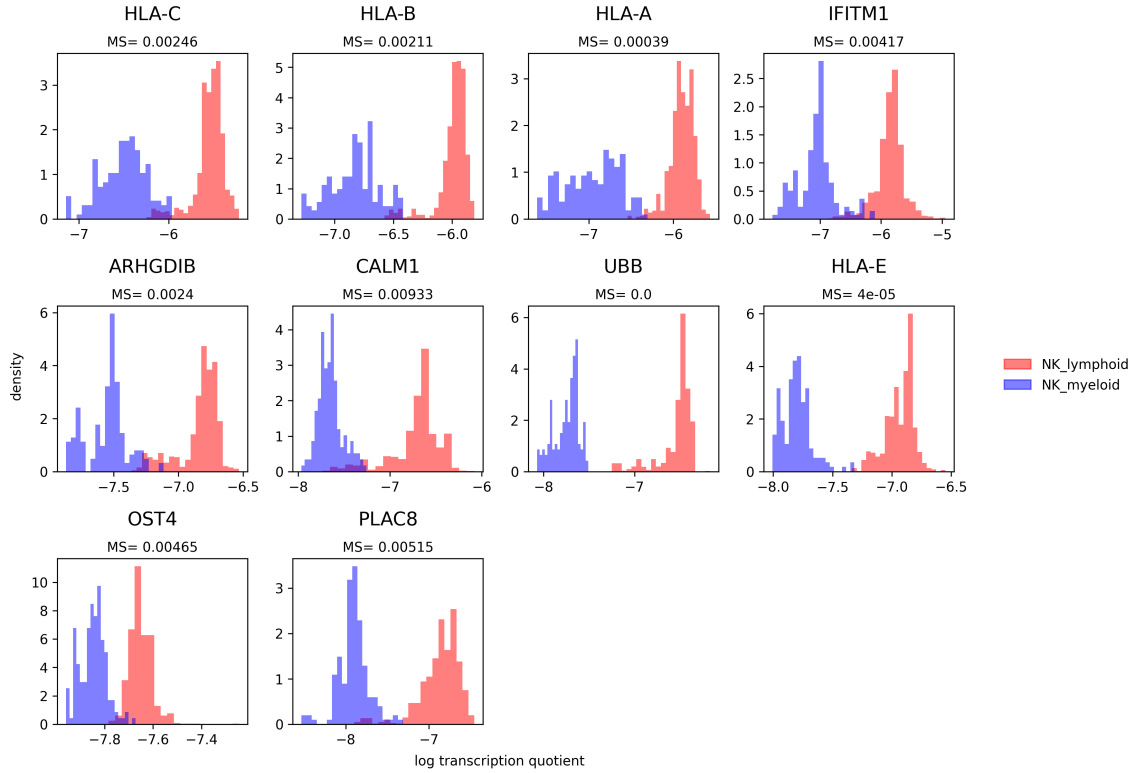


Figure S31. Marker genes that are consistently higher expressed in lymphoid-NK cells than myeloid-NK cells. For the two subpopulations of NK-cells that were identified by the *Bonsai* representation of the blood cell dataset [4] (see Fig 4), we found all marker genes with a marker score below 0.01, i.e., the probability that the gene is higher expressed in a randomly chosen myeloid-NK cell than in a randomly chosen lymphoid-NK cell is less than 0.01 (see Methods). We then selected the 10 marker genes that had the highest mean expression in the lymphoid-NK cells. Each panel corresponds to one marker gene (indicated at the top of the panel together with its marker score). The red and blue histograms show the distribution of expression levels (log transcription quotients) in the 925 lymphoid-NK and 155 myeloid-NK cells, respectively.

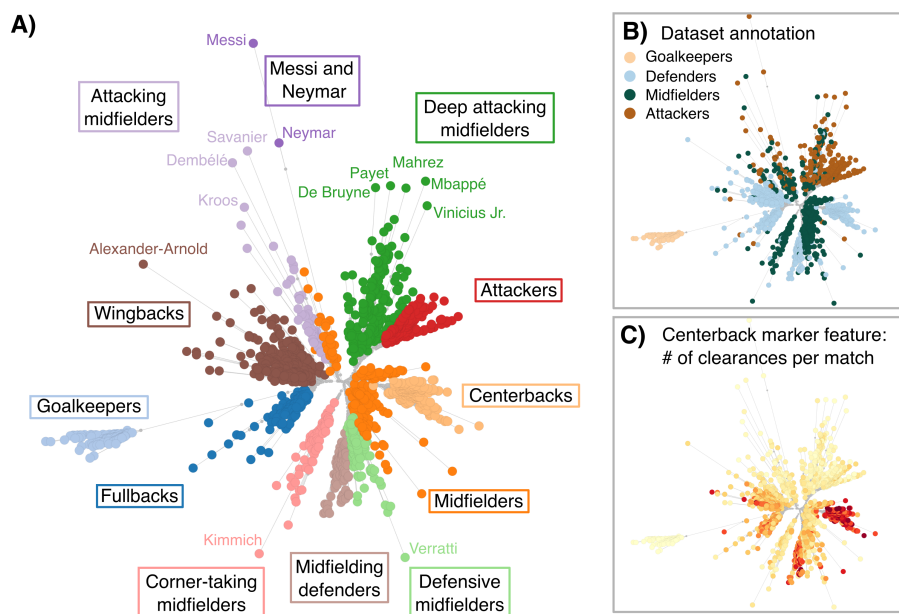


Figure S32. *Bonsai* is applicable to a wide variety of datasets. **A)** The *Bonsai*-tree representing the statistics of 1436 football players from five international competitions in the season 2021-2022 [15]. We preprocessed the data by shifting and normalizing each feature to have zero mean and unit variance and then projecting the data on the first 50 PCA components. The colors indicate 12 clusters of players that were found by our tree-based clustering method (see Methods), which we manually annotated using example players and marker features (see Supp. Figs. S33-S35). In addition, names of the 13 most outlying players are shown. **B)** The same *Bonsai* tree with players colored by their annotated position on the field shows that these clearly cluster on the tree. **C)** An example of a marker feature identified by *Bonsai-scout*: central defenders can be distinguished from the other players based on the number of clearances per match.

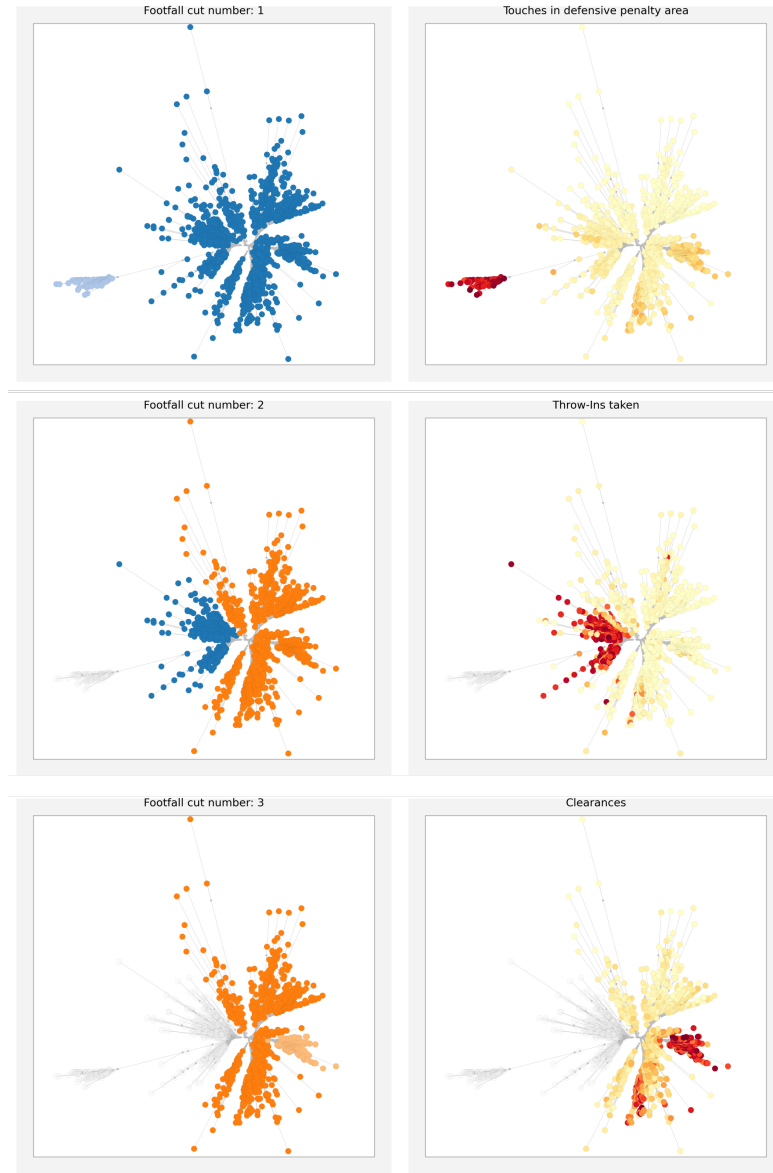


Figure S33. Marker genes distinguishing clusters in the football player dataset. For the dataset with statistics on 1436 football players [15], we found 12 player-clusters using our clustering method that iteratively cuts the tree into smaller subtrees (see Methods). The left panels show, from top to bottom, the first three cuts, and values of example marker features for each split are shown on the right. In the first cut, goalkeepers are split from other players with marker ‘Touches in defensive penalty area’. The second cut splits full backs and wing backs from the others with marker ‘Throw-ins taken’. The third cut splits center backs from the remaining players (i.e., the subtree without the goalkeepers and fullbacks) with marker the number of ‘Clearances’. Thus, after the first three cuts, goal keepers and defenders were split off from the rest.

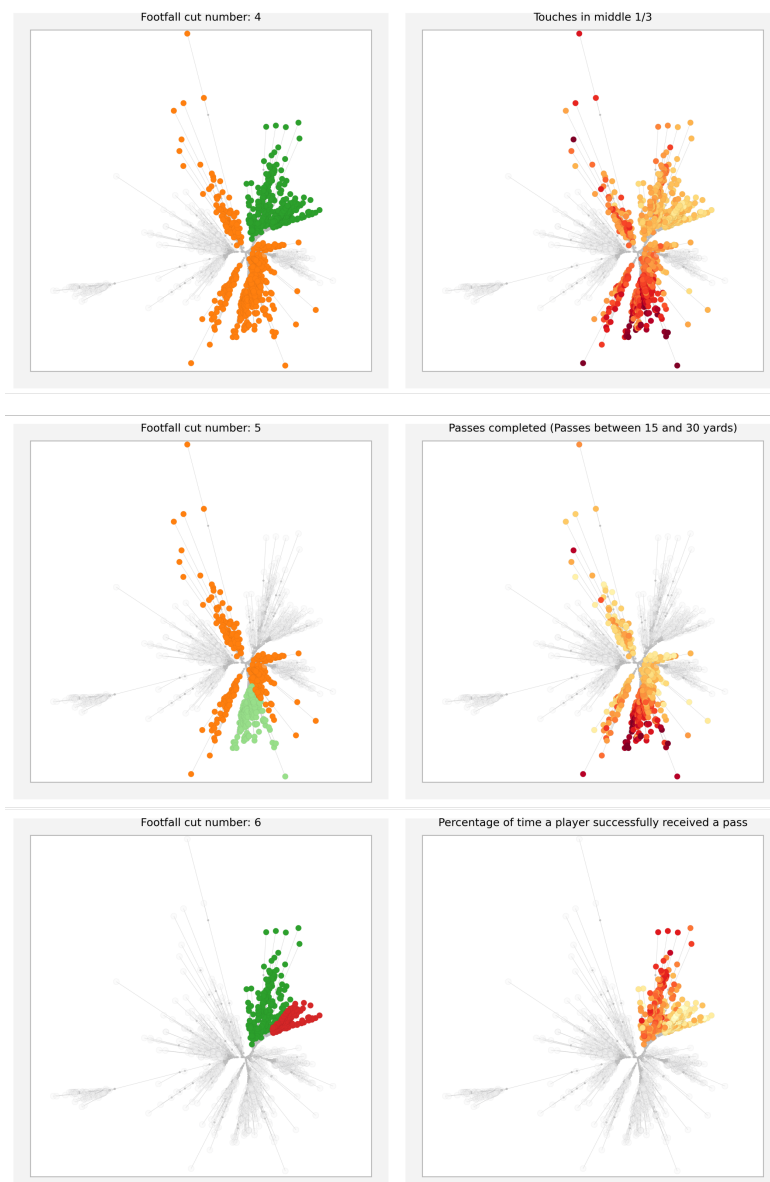


Figure S34. Marker genes distinguishing clusters in the football player dataset. For the dataset with statistics on 1436 football players [15], we found 12 player-clusters using our clustering method that iteratively cuts the tree into smaller subtrees (see Methods). The left panel shows, from top to bottom, cuts 4-6, and values of example marker features for each split are shown on the right. The fourth split cuts attackers from the midfielders with as the marker a lower number of 'Touches in the middle'. The fifth cut splits a group of defensive midfielders and midfielding defenders from the other midfielders using as the marker feature the number of 'Passes between 15 and 30 yards completed'. The sixth cut splits the attackers into deep attacking midfielders and pure attackers with marker that pure attackers have a lower percentage of successfully received passes.



Figure S35. Marker genes distinguishing clusters in the football player dataset. For the dataset with statistics on 1436 football players [15], we found 12 player-clusters using our clustering method that iteratively cuts the tree into smaller subtrees (see the description in the Methods-section). The left panel shows, from top to bottom, cuts 7-9, and values of example marker features for each split are shown on the right. Cut seven splits a group of midfielders from the other midfielders and is characterized by taking corners more than others. The eighth cut splits just two outlier players (Messi and Neymar) from the rest, with marker the number of completed passes that are sent into open space between back defenders of the other team. The ninth cut splits off a group of attacking midfielders that is characterized by having many ‘Shot-creating actions’.



Figure S36. Marker genes distinguishing clusters in the football player dataset. For the dataset with statistics on 1436 football players [15], we found 12 player-clusters using our clustering method that iteratively cuts the tree into smaller subtrees (see the description in the Methods-section). The left panel shows, from top to bottom, cuts 10 and 11, and values of example marker features for each split are shown on the right. The tenth cut splits wing backs from fullbacks, with wing backs being characterized by more often moving the ball forward at least 5 yards. Finally, the eleventh cut splits what we have called midfielding defenders from defensive midfielders, where defensive midfielders are characterized by more often applying pressure to an opposing player that has the ball.

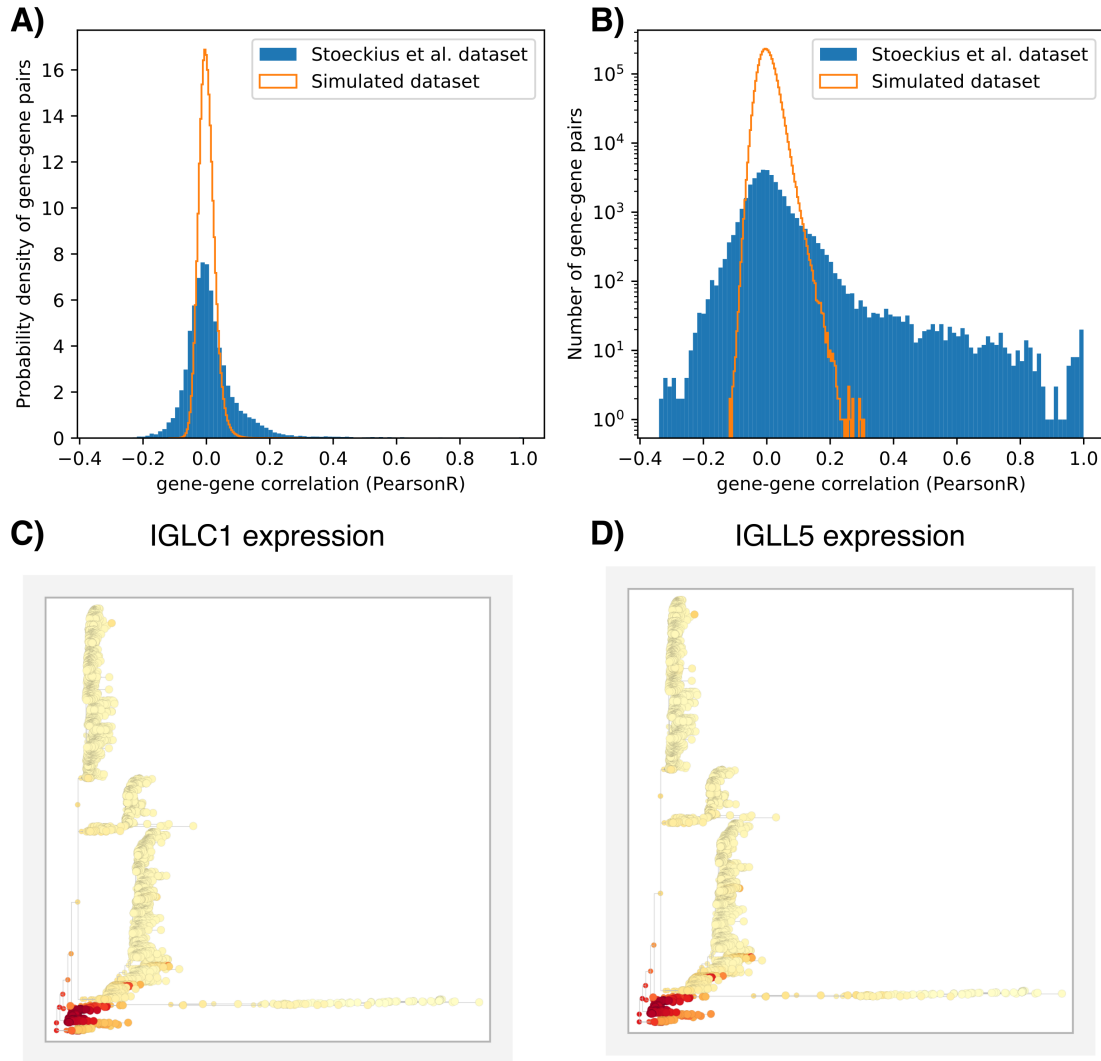


Figure S37. The homogeneous diffusion prior for gene expression changes does not inhibit *Bonsai* from detecting correlations between genes in the data. We can calculate the correlations in the inferred gene expression changes along the branches of the reconstructed *Bonsai*-trees. In **A)**-**B)**, we show these correlations for all gene-gene pairs for two datasets: 1) for the blood dataset by Stoeckius et al. [4] that was discussed in the main text, and 2) for the simulated dataset based on a perfect binary tree. The data in the simulated dataset are purely based on the homogeneous diffusion prior, and indeed we see only small correlations between genes that arise because we take a finite sample of expression changes. However, the real dataset shows much stronger correlations between genes, proving that the homogeneous diffusion prior is easily overruled by the data if there is evidence for the correlated change of genes. We show the same data in **A)** and **B)**, but in **A)** we show the normalized histogram, while in **B)** we show the y-axis on a log-scale. In **C)**-**D)**, we show an example of two strongly correlated genes, which indeed have a very similar expression pattern. Since these genes encode for parts of the immunoglobulins, which are produced by B-cells, it indeed makes sense that these are strongly correlated.

SI.B The likelihood of the data given a tree

In this section we derive the equations for calculating the likelihood of the data given a tree topology, T , and its branch lengths, $\mathbf{t}$: $P(D|T, \mathbf{t})$.

Notational conventions. In the following, we will define a tree by its topology T of a set of nodes, connected by branches with branch lengths $\mathbf{t}$. Even though the likelihood that we will derive is independent of which node we designate as the root, we will for notational convenience require that we have defined which node is the root. For a rooted tree, each branch has a unique upstream node, so that we can conveniently indicate the branch upstream of node i also as branch i with branch length t_i .

The leaf-nodes of the tree correspond to the data points that were provided to the algorithm. Bonsai assumes that for each leaf-node i , the input data provides an estimated position $\boldsymbol{\mu}_i$, i.e., a vector in a continuous d -dimensional ‘feature space’ with a vector of error-bars $\boldsymbol{\sigma}_i$ that correspond to the standard-deviations of the ‘noise’ in the measurement of each feature of node i . Although Bonsai can take any input data of this type, in the following we will use notation and terminology as if the data points are cells, and the estimated ‘features’ μ_{gi} correspond to the log-expression of gene g in cell i , and the error-bars σ_{gi} are the corresponding measurement noise levels. We generally denote the (unknown) ‘true’ positions of each node i as vectors $\mathbf{x}_i$ in the d -dimensional feature space (e.g. gene expression space). Lastly, we will assume that it is clearly defined which observed cell data is supposed to come from which node on the tree and use the same index i for the node corresponding to cell i .

SI.B.1 The likelihood of a tree with given branch lengths

The likelihood of the data D given a certain tree T with branch lengths $\mathbf{t}$ is the product of two types of terms:

1. The likelihood of observing the mRNA-counts for cell i given a true position of that cell in gene expression space. We’ll denote this term by $P(\text{cell}_i|\mathbf{x}_i)$.
2. The likelihood that node i has moved through gene expression space from the position of its ancestor in the tree $\pi(i)$. This term is denoted by $P(\mathbf{x}_i|\mathbf{x}_{\pi(i)}, t_i)$.

Compiling these terms, we get the likelihood of the data *and* all tree node coordinates conditioned on the tree topology and the branch lengths:

$$P(D, \{\mathbf{x}_i\}|T, \mathbf{t}) = \prod_{i \text{ in cells}} P(\text{cell}_i|\mathbf{x}_i) \prod_{\substack{j \in \text{nodes} \\ j \neq \text{root}}} P(\mathbf{x}_j|\mathbf{x}_{\pi(j)}, t_j) P(\mathbf{x}_{\text{root}}),$$

where the $P(\mathbf{x}_{\text{root}})$ is the prior probability for the position of the root. We will take a uniform prior for the position of the root, so that this term will drop out of the likelihood. Based on this, we can get the likelihood of the data conditioned on the tree topology and the branch lengths by marginalizing over the positions of all tree nodes:

$$P(D|T, \mathbf{t}) = \int d\mathbf{x}_1 \cdots \int d\mathbf{x}_n \left[\prod_{i \text{ in cells}} P(\text{cell}_i|\mathbf{x}_i) \prod_{\substack{j \in \text{nodes} \\ j \neq \text{root}}} P(\mathbf{x}_j|\mathbf{x}_{\pi(j)}, t_j) \right], \quad (\text{S1})$$

where n is the number of nodes in the tree. In the following subsections, we will provide more detail on the individual likelihood terms and solve the integrals.

SI.B.1.1 Likelihood of the data at the leaf-nodes given a true position

One part of the likelihood corresponds to factors $P(\text{cell}_i|\mathbf{x}_i)$ for the data of object i given an assumed position $\mathbf{x}_i$ in the feature space. Bonsai assumes that for each object i , the input specifies a vector of estimated position $\boldsymbol{\mu}_i$ with error-bars $\boldsymbol{\sigma}_i$, and that this factor in the likelihood is given by a product of Gaussian distributions:

$$P(\text{cell}_i|\mathbf{x}_i) = \prod_g P(\text{cell}_i|x_{gi}) = \prod_g \frac{1}{\sqrt{2\pi}\sigma_{gi}} \exp\left(-\frac{(x_{gi} - \mu_{gi})^2}{2\sigma_{gi}^2}\right), \quad (\text{S2})$$

That is, we require that the input data is normalized such that the likelihood of the ‘measured’ features $\boldsymbol{\mu}_i$ is reasonably approximated by a multi-variate Gaussian with standard-deviations $\boldsymbol{\sigma}_i$ and negligible covariances. As we discuss for scRNA-seq data below, this might require a careful definition of the variables to be inferred, which will depend on the data type. Apart from the specific application for scRNA-seq, it will be up to the user to provide appropriately normalized data to *Bonsai*.

SI.B.1.2 The likelihood of gene expression data given a cell’s true gene expression state

In scRNA-seq data, the experimental output after preprocessing is a so-called count table, which is a matrix of mRNA-counts for each gene (as rows) and for each cell (as columns). However, to normalize this data into estimated positions in gene expression space μ_{gi} with error-bars σ_{gi} we need to carefully take both technical and biological noise into account [16–19]. For this, we will use *Sanity* [3], a Bayesian method that provides the best guess for a cell’s position in gene expression space *and* uncertainty estimates on these guesses. We refer to the *Sanity*-publication for a detailed description of this method, but we summarize the key points here.

As described in the *Sanity* publication [3], the mRNA counts in a cell are the result of the processes of transcription and mRNA decay, which are intrinsically stochastic due to the unavoidable thermal noise at the molecular level. Consequently, rather than controlling precisely when how many mRNAs of which genes are made or degraded, the physical state of the cell only determines the *rates* at which each gene is transcribed and its mRNAs are degraded. As the physical state of the cell changes in time, these transcription and degradation rates will vary as well. Consequently, the number of mRNAs m_{gc} for gene g in a given cell c is a *stochastic* function of the transcription rate $\lambda_{gc}(t)$ and mRNA degradation rate $\mu_{gc}(t)$ of this gene in the recent past. In particular, the expected number of mRNAs is given by

$$\langle m_{gc} \rangle = \int_0^\infty dt \lambda_{gc}(t) \exp\left(-\int_0^t \mu_{gc}(\tau) d\tau\right) \equiv a_{gc}, \quad (\text{S3})$$

with $\lambda_{gc}(t)$ and $\mu_{gc}(t)$ the transcription and mRNA decay rates a time t in the past of this cell. We denote this expected mRNA counts as the *transcriptional activity* a_{gc} of gene g in cell c . The *actual* number of mRNAs in the cell will be a sample from a Poisson distribution with mean a_{gc} .

To a reasonable approximation, the number of captured mRNA-molecules, n_{gc} , in a scRNA-seq experiment correspond to a random sample of the true mRNA molecule number in the cell c [3, 16, 20].

Consequently, the combined effects of the intrinsic biological and technical noise are that these counts n_{gc} are still Poisson samples of the transcriptional activities a_{gc} [3].

Because the efficiency of capture varies from cell to cell, and because it can be argued that two cells with proportional transcription activities $a_{gc'} = \lambda a_{gc}$ are effectively in the same gene expression state (i.e., they only differ in the absolute size of their mRNA pool), it is more meaningful to describe the gene expression state by *transcription quotients* $\alpha_{gc} = a_{gc}/(\sum_{g'} a_{g'c})$. Finally, because absolute gene expression levels vary over 4 – 5 orders of magnitude between genes, it is generally more meaningful to express gene expression levels in logarithmic scale (so that the same fold-change in expression corresponds to the same change, irrespective of the absolute expression level). Therefore, *Sanity* expresses the gene expression state of the cell in terms of *Log Transcription Quotients* or LTQs:

$$x_{gc} = \log(\alpha_{gc}) = \log\left(\frac{a_{gc}}{\sum_{g'} a_{g'c}}\right), \quad (\text{S4})$$

where the a_{gc} are defined in terms of transcription and mRNA decay rates according to equation (S3).

Sanity uses a Bayesian algorithm to estimate the LTQs, x_{gc} , for each gene g in each cell c . In its output, it reports the means x_{gc}^* and standard-deviations ϵ_{gc} of the *posterior* distributions of the LTQs for each gene in each cell. In addition, it also reports an estimate v_g of the true variance in LTQs of gene g across the cells in the dataset. Notably, because of the Poisson sampling noise, and because absolute expression levels vary over several orders of magnitude, the resulting sizes of the error-bars ϵ_{gc} vary enormously across genes and even across cells for the same gene. This variation in measurement accuracy is crucial to take into account in the analysis of scRNA-seq data.

Finally, while *Sanity* provides posterior estimates, *Bonsai* requires as input estimated positions and error-bars of the *likelihood* instead of the posterior. Fortunately, the posterior is proportional to the product of the desired likelihood term and *Sanity*'s prior, such that the likelihood can be well-approximated from the posterior. We will therefore follow the derivation given in [3] and obtain the measurement estimates μ_{gc} and error-bars σ_{gc} from the reported posterior estimates x_{gc}^* and posterior error-bars ϵ_{gc} using the identities:

$$\begin{aligned} \mu_{gc} &= x_{gc}^* \frac{v_g}{v_g - \epsilon_{gc}^2}, \\ \sigma_{gc}^2 &= \epsilon_{gc}^2 \frac{v_g}{v_g - \epsilon_{gc}^2}, \end{aligned} \quad (\text{S5})$$

where v_g is *Sanity*'s estimate of the variance for gene g .

Note on computational implementation. Practically, we can thus use the Equations (S5) to transform the output from *Sanity*, and then the likelihood term $P(\text{cell}_i | \mathbf{x}_{gi})$ takes the Gaussian shape in μ_{gi} and σ_{gi} as in Equation (S2).

Additionally, to be able to reconstruct this likelihood best, we made a slightly altered version of the *Sanity*-implementation. Normally, *Sanity* returns values that are obtained by integrating over the possible values of v_g , so that the returned v_g is the expectation value of v_g , rather than the maximum posterior estimate. Although this is the most correct option for obtaining posterior estimates of the LTQs, the transformation to reconstruct the likelihood can become inaccurate in

some cases. Therefore, we now adapted *Sanity* to have a mode in which it returns the values for the v_g that maximize the posterior.

SI.B.1.3 Selecting features with signal-to-noise over a threshold

In scRNA-seq, many genes with relatively low expression are sampled only very sparsely, so that the error-bars on their expression levels may be much larger than the true variation in their expression levels across cells. Such genes contain very little information about the similarities of gene expression across cells (and thus the tree structure) and we have found that including them leads to worse performance of *Bonsai* on simulated data. That is, they appear to add more noise than they add information about the structure in the data. We therefore decided to only include genes (or more generally, features) that have signal-to-noise over a threshold.

Let x_{gi}^* denote the posterior estimate of the LTQ of gene g in cell i , x_g the estimate of the mean LTQ over the cells, and ϵ_{gi}^2 the error-bar on the deviation $\delta_{gi} = x_{gi}^* - x_g$. We then define the signal-to-noise ratio, S_g , as the average ratio of the posterior estimate of the squared deviation of the ‘signal’ and the posterior error-bars ϵ_{gi}^2 on this signal, i.e.

$$S_g = \frac{1}{C} \sum_i \frac{\delta_{gi}^2}{\epsilon_{gi}^2}, \quad (\text{S6})$$

with C the number of cells. For scRNA-seq data, *Sanity* reports the means and variances of the posteriors on the gene expression values (see Section SI.B.1.2), so that we can use this output directly to estimate S_g .

For other data, we estimate S_g from the input expression estimates μ_{gi} with error-bars σ_{gi} as follows. If we assume the true expression values x_{gi} are drawn from a Gaussian with variance v_g and mean μ_g (which are both unknown), then the likelihood of the estimates μ_{gi} is given by

$$P(D_g|v_g) = \prod_i \frac{1}{\sqrt{2\pi(v_g + \sigma_{gi}^2)}} \exp \left[-\sum_i \frac{(\mu_{gi} - \mu_g)^2}{2(v_g + \sigma_{gi}^2)} \right]. \quad (\text{S7})$$

We now first estimate the mean, μ_g , by its maximum likelihood value, $\bar{\mu}_g$, which gives

$$\bar{\mu}_g \equiv \sum_i \frac{\mu_{gi}}{v_g + \sigma_{gi}^2} \left[\sum_i \frac{1}{v_g + \sigma_{gi}^2} \right]^{-1}. \quad (\text{S8})$$

Note that this maximum likelihood estimate, $\bar{\mu}_g$, depends on the variance v_g that we still need to be estimate. We therefore substitute the true mean, μ_g , by the estimate, $\bar{\mu}_g$, in the likelihood given by Equation (S7), and next estimate the variance v_g by maximizing this likelihood, $P(D_g|v_g)$. The maximal likelihood value of v_g is a solution of the equation

$$\sum_i \frac{1}{v_g + \sigma_{gi}^2} \left[\frac{(\mu_{gi} - \bar{\mu}_g)^2}{v_g + \sigma_{gi}^2} - 1 \right] = 0. \quad (\text{S9})$$

Given the estimated value of the variance v_g , the posterior estimate of the ‘signal’ ($x_{gi} - x_g$) in cell i is given by

$$\delta_{gi} = \langle (x_{gi} - x_g) \rangle = \frac{v_g}{v_g + \sigma_{gi}^2} (\mu_{gi} - \bar{\mu}_g), \quad (\text{S10})$$

and the error-bar on this estimate is

$$\epsilon_{gi}^2 = \frac{v_g \sigma_{gi}^2}{v_g + \sigma_{gi}^2}. \quad (\text{S11})$$

We then finally estimate the signal-to-noise S_g using equation (S6).

By default we select all genes with $S_g \geq 1$ to include in the *Bonsai* analysis. However, if desired, users can change the threshold value on S_g .

SI.B.1.4 The prior probability of movement in gene expression state

The second term in the tree likelihood (Equation (S1)) is the prior probability of a movement in gene expression space from a position $\mathbf{x}_{\pi(i)}$ to the position $\mathbf{x}_i$. This term captures how likely the inferred gene expression changes along the branches of the tree are.

It is important to note that $P(\mathbf{x}_i|\mathbf{x}_{\pi(i)})$ captures only the *prior* probability distribution on gene expression changes, while by reconstructing the tree on the data we try to infer the *posterior* distribution of gene expression changes. Since we do not want to impose strong assumptions on which gene expression changes are more likely than others, we want to take a very conservative prior. Therefore, we only assume that 1) gene expression changes can be described by a continuous Markov process, and 2) gene expression changes are *a priori* equally likely in all genes relative to the total variation in that gene in the dataset.

The assumption that we can describe gene expression changes by a continuous Markov process means that we expect no discontinuous jumps in gene expression, and that we expect the current gene expression state to be predictive of changes in gene expression.¹ In [21, 22], it is shown that, under a mild additional assumption², any continuous Markov process must be a drift-diffusion process, which can be described by the form

$$\partial_t P(\mathbf{x}_i, t | \mathbf{x}_{\pi(i)}, t') = - \sum_g \frac{\partial}{\partial x_{gi}} [A_g(\mathbf{x}_i, t) P(\mathbf{x}_i, t | \mathbf{x}_{\pi(i)}, t')] + \sum_{g, g'} \frac{1}{2} \frac{\partial^2}{\partial x_{gi} \partial x_{g'i}} [B_{gg'}(\mathbf{x}_i, t) P(\mathbf{x}_i, t | \mathbf{x}_{\pi(i)}, t')].$$

This equation describes the time evolution of the probability distribution of a cell to be at gene expression state $\mathbf{x}_i$ at time t while it was at $\mathbf{x}_{\pi(i)}$ at timepoint t' . The term A_g describes a deterministic drift that drives the cell's expression and the term $B_{gg'}$ describes stochastic diffusion. In our prior process, we will set A_g to zero so as not to assume any deterministic drift; rather, we aim to learn gene expression changes from the data instead of imposing uncertain prior knowledge. Similarly, for the diffusion term $B_{gg'}$, we do not want to assume prior knowledge about any correlation between the genes, so that we will assume independent diffusion across the genes: $B_{gg'} = B_g \delta_{gg'}$. Finally, we set B_g such that the amount of expected diffusion in a gene is proportional to the total variation that is observed in the dataset: $B_g = v_g$. We thus get

$$\partial_t P(\mathbf{x}_i, t | \mathbf{x}_{\pi(i)}, t') = \sum_g \frac{1}{2} \frac{\partial^2}{\partial x_g^2} v_g P(\mathbf{z}, t | \mathbf{y}, t'),$$

¹Note that this assumption still allows for a change in gene expression to be caused by a signal outside of the cell, as long as the change can be predicted from the current gene expression. For instance, a spatial signal may drive changes in gene expression, but as long as the cell's spatial position is reflected in its current expression then the current state may still be predictive of the changes.

²The additional assumption is that the probability distribution of gene expression changes over a very short time interval δt has a finite mean and variance that scale proportionally to δt .

of which the solution is given by a Gaussian

$$P(\mathbf{x}_i, t | \mathbf{x}_{\pi(i)}, t') = \prod_g \frac{1}{\sqrt{2\pi v_g(t-t')}} \exp\left(-\frac{(x_{gi} - x_{g\pi(i)})^2}{2v_g(t-t')}\right). \quad (\text{S12})$$

This gives us the likelihood term for movement in gene expression space where the diffusion time along the branch $t_i := t - t'$.

SI.B.1.5 The full tree likelihood expression

Taking together Equations (S1), (S2), and (S12), we get an expression for the likelihood of the tree:

$$P(D|T, \mathbf{t}) = \prod_g \int dx_{g1} \cdots \int dx_{gn} \left[\prod_{i \text{ in cells}} \frac{1}{\sqrt{2\pi\sigma_{gi}}} \exp\left(-\frac{(x_{gi} - \mu_{gi})^2}{2\sigma_{gi}^2}\right) \right. \\ \left. \times \prod_{\substack{j \in \text{nodes} \\ j \neq \text{root}}} \frac{1}{\sqrt{2\pi v_g t_j}} \exp\left(-\frac{(x_{gj} - x_{g\pi(j)})^2}{2v_g t_j}\right) \right]. \quad (\text{S13})$$

It is important to note here that the set of cells is a subset of the set of nodes. Every cell i has an associated node with coordinates x_{gi} that will show up in the first term, but these will also show up in the second term.

Practical note: the likelihood is independent on the choice of root. Because we took a uniform prior over the root position, and because the expression for the likelihood of movement is symmetric in $\mathbf{x}_j$ and $\mathbf{x}_{\pi(j)}$, we have the following identity

$$P(\mathbf{x}_j | \mathbf{x}_{\pi(j)}, t_j) P(\mathbf{x}_{\pi(j)}) = P(\mathbf{x}_{\pi(j)} | \mathbf{x}_j, t_j) P(\mathbf{x}_j). \quad (\text{S14})$$

As a result, we can always designate another node as the root without affecting the tree likelihood. We will use this freedom to choose the root node in several derivations and computational implementations.

SI.B.1.6 The tree likelihood for other types of data

Although we have focused the discussion above on single-cell RNA-sequencing data, the equations for the tree likelihood generalize to other types of data. Importantly, the assumptions about the data that we have made above should hold, in particular

- We should be able to ‘normalize’ the data such that each cell or sample can be described by coordinates x_{gi} , and the likelihood of the data can be well-described by a multivariate Gaussian distribution in these coordinates with negligible covariance (analogous to Eq. (S2)).
- The prior likelihood of moving from $\mathbf{x}_{\pi(j)}$ to $\mathbf{x}_j$ in these coordinates can be described by a Markov process in which we do not expect discontinuous jumps.

Once these assumptions are met, the rest of the derivations presented here hold for any data type. To facilitate a clean presentation, we will in the following borrow *scRNA-seq* terminology (e.g., *gene*, *cell*, *LTQ*), and we leave it to the reader to translate these terms to analogs for their preferred data types.

SI.B.2 Analytical solutions for the tree likelihood

In this section, we will analytically solve the integrals from Equation (S13). To facilitate this derivation we will introduce several variable transformations, and I will give an overview of those here.

Definition of variables used throughout this SI

Tree variables

- t_i : the branch length (or diffusion time) of the branch upstream of node i .
- v_g : the estimated total variance for gene g across the dataset
- x_{gk} : the true LTQ of gene g in node k . If the node-label k matches a cell-label i , this cell is associated with that node.

Observed cell variables

- μ_{gi} : the mean of the Gaussian distribution describing the likelihood of the experimental data given the log-transcription quotient (LTQ) of gene g in cell i . This mean is only defined for cell-associated nodes.
- σ_{gi}^2 : the corresponding variance of the likelihood distribution.
- $w_{gi} = \frac{1}{\sigma_{gi}^2}$: the *precision* for gene g at cell i , defined as the inverse of the variance.
- w_{gi}^* : the *precision* for gene g at node i when corrected for the uncertainty induced by diffusing for a time t_i .

Effective leaf variables

- $\bar{\mu}_{gk}$: the mean of the Gaussian distribution that describes the likelihood for the entire subtree downstream of node k . This mean is thus also defined for nodes not associated with a cell. *Note that these effective leaf variables depend on the definition of the root since this determines which nodes are downstream. If this is not clear from the context this will be indicated by $\bar{\mu}_{gk}(r)$, where r is the designated root.*
- $\bar{w}_{gk}$: the corresponding precision for the effective leaf. The combination $\bar{w}_{gk}^*$ indicates the precision when accounted for the uncertainty induced by diffusion.
- $C_r(a)$: the set of leaves that are directly downstream of node a , conditioned on r being designated as the root.

SI.B.2.1 Solving the integral for the star-tree

We will first solve the integrals in Equation (S13) in the simplified case of a *star-tree*, which is the simple tree where all cells have an associated node and these ‘cell-nodes’ are all connected to a single root. Denoting the number of cells by $|C|$, the star-tree thus has $|C| + 1$ nodes and $|C|$ edges. Later, we will show that we can use the star-tree result to solve the integrals for any tree. Denoting the coordinates of the root-node by x_{gr} , the equations for the star-tree likelihood become

$$P(D|T_{\text{star}}, \mathbf{t}) = \prod_g \left[\int dx_{gr} \prod_i \int dx_{gi} \frac{1}{\sqrt{2\pi v_g t_i}} \exp\left(-\frac{(x_{gi} - x_{gr})^2}{2v_g t_i}\right) \frac{1}{\sqrt{2\pi}\sigma_{gi}} \exp\left(-\frac{(x_{gi} - \mu_{gi})^2}{2\sigma_{gi}^2}\right) \right]. \quad (\text{S15})$$

The integral over x_{gi} can be done by completing the squares in the exponent and recognizing the standard Gaussian shape. This gives

$$P(D|T_{\text{star}}, \mathbf{t}) = \prod_g \int dx_{gr} \prod_i (2\pi(t_i v_g + \sigma_{gi}^2))^{-1/2} \exp\left[-\sum_g \frac{(\mu_{gi} - x_{gr})^2}{2(t_i v_g + \sigma_{gi}^2)}\right], \quad (\text{S16})$$

It is useful to here introduce the ‘precision’ of a node, $w_{gi} = 1/\sigma_{gi}^2$, which is defined as the inverse of the variance in the posterior for the node position. We also define w_{gi}^* as the precision corrected for the uncertainty induced by diffusion over a time t_i : $w_{gi}^* := 1/(t_i v_g + \sigma_{gi}^2)$. We can then write the likelihood of the star-tree in terms of the distances from all the leaves to a center of mass $\bar{\mu}_{gr} = \frac{\sum_i w_{gi}^* \mu_{gi}}{\bar{w}_{gr}}$ where $\bar{w}_{gr} := \sum_j w_{gj}^*$. We will derive this per gene first, and only afterwards take the product. Therefore, we drop the subscripts g for now:

$$P(D|T_{\text{star}}, \mathbf{t}) = \int dx_r \prod_i \left(\frac{w_i^*}{2\pi}\right)^{1/2} \exp\left(-\frac{1}{2} \sum_i w_i^* (\mu_i - x_r)^2\right).$$

Completing the squares in the exponent and re-writing gives

$$\begin{aligned} \sum_i w_i^* (\mu_i - x_r)^2 &= \bar{w}_r (x_r - \bar{\mu}_r)^2 - \bar{w}_r \bar{\mu}_r^2 + \sum_i w_i^* \mu_i^2 \\ &= \bar{w}_r (x_r - \bar{\mu}_r)^2 + \bar{w}_r \bar{\mu}_r^2 - 2\bar{w}_r \bar{\mu}_r^2 + \sum_i w_i^* \mu_i^2 \\ &= \bar{w}_r (x_r - \bar{\mu}_r)^2 + \sum_i w_i^* \bar{\mu}_r^2 - 2\bar{\mu}_r \sum_i w_i^* \mu_i + \sum_i w_i^* \mu_i^2 \\ &= \bar{w}_r (x_r - \bar{\mu}_r)^2 + \sum_i w_i^* (\bar{\mu}_r - \mu_i)^2, \end{aligned}$$

and inserting this gives

$$\begin{aligned}
P(D|T_{\text{star}}, \mathbf{t}) &= \int dx_r \prod_i \left(\frac{w_i^*}{2\pi} \right)^{1/2} \exp \left[-\frac{\bar{w}_r}{2} (x_r - \bar{\mu}_r)^2 - \frac{1}{2} \sum_i w_i^* (\bar{\mu}_r - \mu_i)^2 \right] \\
&= (2\pi)^{-(|C|-1)/2} \bar{w}_r^{-1/2} \prod_i (w_i^*)^{1/2} \exp \left[-\frac{1}{2} \sum_i w_i^* (\bar{\mu}_r - \mu_i)^2 \right] \\
&= \left(\frac{\prod_i w_i^*}{(2\pi)^{|C|-1} \bar{w}_r} \right)^{1/2} \exp \left[-\frac{1}{2} \sum_i w_i^* (\bar{\mu}_r - \mu_i)^2 \right],
\end{aligned}$$

For all genes together this becomes

$$P(D|T_{\text{star}}, \mathbf{t}) = (2\pi)^{-G(|C|-1)/2} \prod_g \left(\frac{\prod_i w_{gi}^*}{\bar{w}_{gr}} \right)^{1/2} \exp \left[-\frac{1}{2} \sum_g \sum_i w_{gi}^* (\bar{\mu}_{gr} - \mu_{gi})^2 \right]. \quad (\text{S17})$$

Note that the dependence on the branch lengths t_i enters through:

$$\begin{aligned}
\bar{\mu}_{gr} &= \frac{\sum_i w_{gi}^* \mu_{gi}}{\bar{w}_{gr}}, \\
\bar{w}_{gr} &= \sum_{i \in C_r(r)} w_{gi}^*, \\
w_{gi}^* &= \frac{1}{t_i v_g + \sigma_{gi}^2},
\end{aligned}$$

where we use the notation $C_r(r)$ to indicate the nodes directly downstream of r when the root is r .

SI.B.2.2 Summarizing a sub-tree in an *effective leaf*

We will here show that the contribution to the tree likelihood of a subtree consisting of one ancestor (t_a) with several leaves as children, can be summarized by replacing this sub-tree by a single *effective leaf*. This can be used recursively to reduce any tree to a simple star-tree consisting of a root connected to effective leaves that summarize the different sub-trees. This provides an efficient way of calculating the tree likelihood.

For deriving this, we will focus on the set of leaves downstream of an ancestor that we will label a . Without loss of generality, we can designate the node upstream of a as the root (r), because the tree likelihood is independent of the choice of root. In the following, we will denote by $C_r(a)$ the set of children of ancestor a when r is considered the root. The sub-tree downstream of a we will denote by T_{ds} , and R will be the set of all nodes other than r , a , and the nodes in T_{ds} . The likelihood contribution of the nodes in R only depends on the position of the root r , such that we

can gather this contribution in a term $P(\{z \in R\}|x_{gr})$. We get

$$P(D|T, \mathbf{t}) = \prod_g \int dx_{gr} \left[P(\{z \in R\}|x_{gr}) \right. \\ \left. \times \int dx_{ga} \sqrt{\frac{1}{2\pi t_a v_g}} \exp\left(-\frac{1}{2} \frac{(x_{gr} - x_{ga})^2}{t_a v_g}\right) \prod_{k \in C_r(a)} \sqrt{\frac{w_{gk}^*}{2\pi}} \exp\left(-\frac{1}{2} \sum_{k \in C_r(a)} w_{gk}^* (x_{ga} - \mu_{gk})^2\right) \right], \quad (\text{S18})$$

where the integral over the leaf positions x_{gk} was already performed. We will again do the derivation for one gene at a time, so that we can drop the corresponding subscripts. As before, we can complete the squares in the second exponent. We will denote the effective precision of the effective leaf that summarizes the subtree downstream of a as $\bar{w}_a := \sum_{k \in C_r(a)} w_k^*$, and the effective mean as $\bar{\mu}_a := \frac{\sum_{k \in C_r(a)} w_k^* \mu_k}{\bar{w}_a}$, we get

$$P(D|T, \mathbf{t}) = \int dx_r \left[P(\{z \in R\}|x_r) \int dx_a \left[\sqrt{\frac{1}{2\pi t_a v}} \exp\left(-\frac{1}{2} \frac{(x_r - x_a)^2}{t_a v}\right) \right. \right. \\ \left. \left. \times \sqrt{\frac{\bar{w}_a}{2\pi}} \exp\left(-\frac{1}{2} \bar{w}_a (x_a - \bar{\mu}_a)^2\right) \sqrt{\frac{\prod_{k \in C_r(a)} w_k^*}{(2\pi)^{|C_r(a)|-1} \bar{w}_a}} \exp\left(-\frac{1}{2} \sum_{k \in C_r(a)} w_k^* (\bar{\mu}_a - \mu_k)^2\right) \right] \right].$$

Performing the integral over x_a , we get

$$P(D|T, \mathbf{t}) = \int dx_r \left[P(\{z \in R\}|x_r) \sqrt{\frac{\bar{w}_a}{2\pi(1 + t_a v \bar{w}_a)}} \exp\left(-\frac{1}{2} \frac{\bar{w}_a}{1 + t_a v \bar{w}_a} (x_r - \bar{\mu}_a)^2\right) \right. \\ \left. \times \sqrt{\frac{\prod_{k \in C_r(a)} w_k^*}{(2\pi)^{|C_r(a)|-1} \bar{w}_a}} \exp\left(-\frac{1}{2} \sum_{k \in C_r(a)} w_k^* (\bar{\mu}_a - \mu_k)^2\right) \right].$$

We can see that the second line in this expression only depends on the sub-tree below a and not on the root position x_r . In fact, it is exactly what one would get for the likelihood of a star-tree built from the leaves in $C_r(a)$. We can expose this by writing the likelihood in a form that shows that the subtree downstream of a is now just summarized by an effective leaf with position $\bar{\mu}_a$ and an effective precision $\bar{w}_a$. Here, we recognize that this effective precision becomes $\bar{w}_{ga}^* = \frac{\bar{w}_a}{1 + t_a v \bar{w}_a}$ when accounting for the uncertainty induced by diffusion to node a . For easy reference, we here also

write out all the definitions we used and re-introduce the product over genes:

$$\begin{aligned}
P(D|T, \mathbf{t}) &= P(D|T_{ds}(a), \mathbf{t}) \prod_g \int dx_{gr} \sqrt{\frac{\bar{w}_{ga}^*}{2\pi}} \exp\left(-\frac{1}{2} \bar{w}_{ga}^* (x_{gr} - \bar{\mu}_{ga})^2\right) P(\{z \in R\} | x_{gr}) \\
P(D|T_{ds}(a), \mathbf{t}) &= \sqrt{\prod_g \frac{\prod_{k \in C_r(a)} w_{gk}^*}{(2\pi)^{|C_r(a)|-1} \bar{w}_{ga}}} \exp\left(-\frac{1}{2} \sum_g \sum_{k \in C_r(a)} w_{gk}^* (\mu_{gk} - \bar{\mu}_{ga})^2\right) \\
\bar{w}_{ga}^* &= \frac{\bar{w}_{ga}}{1 + t_a v_g \bar{w}_{ga}} \\
\bar{\mu}_{ga} &= \frac{\sum_{k \in C_r(a)} w_{gk}^* \mu_{gk}}{\bar{w}_{ga}} \\
\bar{w}_{ga} &= \sum_{k \in C_r(a)} w_{gk}^*
\end{aligned} \tag{S19}$$

In these expressions, we have denoted by $P(D|T_{ds}(a), \mathbf{t})$ the likelihood that we would get if the sub-tree downstream of a would be the full tree.

SI.B.2.3 The loglikelihood of a general tree.

Equation (S19) shows that we can write the likelihood of a general tree in terms of the likelihood of smaller sub-trees. This means we can now perform all the integrals for a general tree recursively. It is useful to here collect the general expression for the resulting loglikelihood:

$$\begin{aligned}
L(D|T, \mathbf{t}) &= \sum_{k \in C_r(r)} L(D|T_{ds}(k), \mathbf{t}) - \frac{1}{2} (|G||C_r(r)| - 1) \log(2\pi) \\
&\quad + \frac{1}{2} \sum_g \left[\sum_{k \in C_r(r)} \log(\bar{w}_{gk}^*) - \log\left(\sum_{k \in C_r(r)} \bar{w}_{gk}^*\right) - \sum_{k \in C_r(r)} \bar{w}_{gk}^* (\bar{\mu}_{gr} - \bar{\mu}_{gk})^2 \right] \\
L(D|T_{ds}(k), \mathbf{t}) &= \sum_{m \in C_r(k)} L(D|T_{ds}(m), \mathbf{t}) - \frac{1}{2} (|G||C_k(r)| - 1) \log(2\pi) \\
&\quad + \frac{1}{2} \sum_g \left[\sum_{m \in C_r(k)} \log(\bar{w}_{gm}^*) - \log\left(\sum_{m \in C_r(k)} \bar{w}_{gm}^*\right) - \sum_{m \in C_r(k)} \bar{w}_{gm}^* (\bar{\mu}_{gk} - \bar{\mu}_{gm})^2 \right]
\end{aligned} \tag{S20}$$

In these equations we have written $\bar{w}_{gm}^*$, but it should be understood that if m is a cell-associated node, this should be replaced by w_{gm}^* .

Note on the factors of 2π . By doing the accounting correctly, one can see that the number of factors of $-\frac{1}{2}|G|\log(2\pi)$ will always be exactly the total number of observed cells minus one. This is thus independent of the tree topology, and we can (and will) therefore leave out these factors in the following derivations and in the computational implementation.

Note on the influence of the gene variances v_g . Recall that the diffusion rate for different genes in the diffusion prior was scaled to the observed variance in that gene in the dataset, i.e., the variance in the Gaussian prior for movement in gene expression space was $t_i v_g$. We can now see how these variances influence the loglikelihood. The v_g enter the expression through the effective precisions $\bar{w}_{gk}^*$ and w_{gc}^* . These effective precisions take two different forms:

- for nodes associated with an observed cell we get: $w_{gc}^* = \frac{1}{t_c v_g + \sigma_{gc}^2}$.
- for internal nodes we get: $\bar{w}_{ga}^* = \frac{\sum_{k \in C_r(a)} \bar{w}_{gk}^*}{1 + t_k v_g \sum_{k \in C_r(a)} \bar{w}_{gk}^*}$, where $\bar{w}_{gk}$ are the effective precisions of the child-nodes.

It is now useful to perform the following transformation of the means and variances for the observed cells, i.e., of the raw input that *Bonsai* uses:

$$\tilde{\sigma}_{gc} = \frac{\sigma_{gc}}{\sqrt{v_g}} \quad \text{and} \quad \tilde{\mu}_{gc} = \frac{\mu_{gc}}{\sqrt{v_g}}. \quad (\text{S21})$$

As a consequence, we get ³

- for cell-associated nodes: $w_{gc}^* = \frac{1}{t_c v_g + \sigma_{gc}^2} = \frac{1}{v_g} \frac{1}{t_c + (\tilde{\sigma}_{gc})^2} = \frac{1}{v_g} \tilde{w}_{gc}^*$.
- for internal nodes: $\bar{w}_{ga}^* = \frac{\sum_{k \in C_r(a)} \bar{w}_{gk}^*}{1 + t_k v_g \sum_{k \in C_r(a)} \bar{w}_{gk}^*} = \frac{\frac{1}{v_g} \sum_{k \in C_r(a)} \tilde{w}_{gk}^*}{1 + t_k \sum_{k \in C_r(a)} \tilde{w}_{gk}^*} = \frac{1}{v_g} \tilde{\bar{w}}_{ga}^*$.

As a consequence, for the loglikelihood we get

$$\begin{aligned} L(D|T, \mathbf{t}) &= \sum_{k \in C_r(r)} L(D|T_{ds}(k), \mathbf{t}) + \frac{1}{2} \sum_g \left[\sum_{k \in C_r(r)} \log(\bar{w}_{gk}^*) - \log \left(\sum_{k \in C_r(r)} \bar{w}_{gk}^* \right) - \sum_{k \in C_r(r)} \bar{w}_{gk}^* (\bar{\mu}_{gr} - \bar{\mu}_{gk})^2 \right], \\ &= \sum_k L(D|T_{ds}(k), \mathbf{t}) - \frac{|C_r(r) - 1|}{2} \sum_g \log(v_g) + \frac{1}{2} \sum_g \left[\sum_k \log(\tilde{\bar{w}}_{gk}^*) - \log \left(\sum_k \tilde{\bar{w}}_{gk}^* \right) - \sum_k \tilde{\bar{w}}_{gk}^* (\tilde{\mu}_{gr} - \tilde{\mu}_{gk})^2 \right], \end{aligned}$$

and similarly for the sub-tree likelihoods $L(D|T_{ds}(k), \mathbf{t})$. Just as for the factors of 2π we see that the variances will add a term to the loglikelihood that is independent of the tree topology (only scaling with the number of cells). We can thus, without loss of generality, perform this variable transformation and after that discard the v_g -terms, which is what we will use in the following.

Choosing a new root. We have pointed out before that the tree likelihood is independent of the choice of root. This means that the recursive expression for the loglikelihood (Eq. (S20)) can be written around any internal node. For this, it is important to realize that the definitions of the effective mean and precision of node i ($\bar{w}_{gi}$, $\bar{\mu}_{gi}$) depend on which set of nodes is downstream of this node, which depends on the choice of root. When choosing a new root, we can use that the new means and precisions can be efficiently computed from the old means and precisions.

³It is timely to here apologize for the combined usage of a bar and a tilde in this notation.

SI.B.3 Optimization of the branch lengths

To learn the diffusion times, or branch lengths, that maximize the likelihood of the data, we will optimize the branch lengths, $\mathbf{t}$. For this optimization, it works best to use a quasi-Newton method that uses both the loglikelihood and its derivatives with respect to all branch lengths. For this, it is essential to have a fast way to calculate this loglikelihood and its gradient.

In section SI.B.2.2 we derived that we can summarize the likelihood contribution of any sub-tree by replacing it by an effective leaf. We will now use this to calculate the derivative of the loglikelihood with respect to one of the branch lengths, t_k . We start by picking the corresponding edge (let's say it connects nodes l and k) and we summarize the sub-trees downstream of l as seen from node k , and vice versa. Here, we can denote the nodes neighboring k without node l as $C_l(k)$, and the converse by $C_k(l)$. The likelihood (again for one gene) becomes:

$$P(D|T, \mathbf{t}) = \left(\prod_{m \in C_l(k) \cup C_k(l)} P(D|T_{ds}(m), \mathbf{t}) \right) \int dx_k dx_l \frac{1}{\sqrt{2\pi t_k}} \exp\left(-\frac{(x_k - x_l)^2}{2t_k}\right) \times \sqrt{\frac{\bar{w}_k}{2\pi}} \exp\left(-\frac{1}{2}\bar{w}_k(x_k - \bar{\mu}_k)^2\right) \sqrt{\frac{\bar{w}_l}{2\pi}} \exp\left(-\frac{1}{2}\bar{w}_l(x_l - \bar{\mu}_l)^2\right) \quad (\text{S22})$$

Then performing the integrals, we get

$$P(D|T, \mathbf{t}) = \left(\prod_{m \in C_l(k) \cup C_k(l)} P(D|T_{ds}(m), \mathbf{t}) \right) \frac{1}{\sqrt{2\pi(t_k + \frac{1}{\bar{w}_k} + \frac{1}{\bar{w}_l})}} \exp\left(-\frac{(\bar{\mu}_k - \bar{\mu}_l)^2}{2(t_k + \frac{1}{\bar{w}_k} + \frac{1}{\bar{w}_l})}\right), \quad (\text{S23})$$

resulting in an alternative expression for the loglikelihood (up to factors of 2π):

$$L(D|T, \mathbf{t}) = \left(\sum_{m \in C_l(k) \cup C_k(l)} L(D|T_{ds}(m), \mathbf{t}) \right) - \frac{1}{2} \log\left(t_k + \frac{1}{\bar{w}_k} + \frac{1}{\bar{w}_l}\right) - \frac{1}{2} \left(\frac{(\bar{\mu}_k - \bar{\mu}_l)^2}{t_k + \frac{1}{\bar{w}_k} + \frac{1}{\bar{w}_l}} \right). \quad (\text{S24})$$

In this form, we can easily find the derivative w.r.t. t_k :

$$\frac{\partial L(D|T, \mathbf{t})}{\partial t_k} = -\frac{1}{2} \frac{1}{t_k + \frac{1}{\bar{w}_k} + \frac{1}{\bar{w}_l}} \left(1 - \frac{(\bar{\mu}_k - \bar{\mu}_l)^2}{t_k + \frac{1}{\bar{w}_k} + \frac{1}{\bar{w}_l}} \right). \quad (\text{S25})$$

This can be done for all branch lengths in the tree to get the full derivative.

SI.B.4 Optimally placing a new cell on an existing tree

During the tree reconstruction, we will at two points need a method for attaching a new cell to an existing tree such that the resulting tree likelihood is maximized: in the SPR-moves (see SI.C.5), and in *Backbone-based-Bonsai* (see Methods). Here, we will therefore first derive an expression for the change in likelihood when cell n with coordinates μ_{gn} and uncertainty σ_{gn}^2 is added to node k with branch length t_n . We will again use that we can summarize the likelihood contribution of any sub-tree by replacing it by an effective leaf, in this case by viewing the entire existing tree as a

sub-tree downstream of node k . Learning from Equation (S24), we get (for each gene)

$$L(D|T \cup n, \mathbf{t}) = \left(\sum_{m \in C_n(k)} L(D|T_{ds}(m), \mathbf{t}) \right) - \frac{1}{2} \log \left(t_n + \frac{1}{\bar{w}_k} + \frac{1}{w_n} \right) - \frac{1}{2} \left(\frac{(\bar{\mu}_k - \mu_n)^2}{t_n + \frac{1}{\bar{w}_k} + \frac{1}{w_n}} \right), \quad (\text{S26})$$

while if cell n is not attached to the tree, we would only get the first term

$$L(D|T, \mathbf{t}) = \left(\sum_{m \in C_n(k)} L(D|T_{ds}(m), \mathbf{t}) \right).$$

The loglikelihood change induced by adding cell n downstream of node k with branch length t_n is thus

$$dL = -\frac{1}{2} \log \left(t + \frac{1}{\bar{w}_k} + \frac{1}{w_n} \right) - \frac{1}{2} \left(\frac{(\bar{\mu}_k - \mu_n)^2}{t_n + \frac{1}{\bar{w}_k} + \frac{1}{w_n}} \right). \quad (\text{S27})$$

Note that the derivative w.r.t. the branch length t_n is the same as in Equation (S25), which we can use for numerically optimizing the length of this attachment-branch.

SI.B.4.1 Strategies for finding the optimal attachment node

Now that we can calculate the loglikelihood change of adding cell n to a specific node in the existing tree, we can start our search for the optimal attachment node. In the *Bonsai*-implementation, we have added three possible strategies. First, one can simply exhaustively calculate the attachment likelihood-change for all nodes in the existing tree and pick the best node. Clearly, however, this is not feasible for large datasets if many cells need to be added.

Second, we can start at the root of the existing tree, and follow the tree-structure by always following the edge that leads to the neighbor with the best attachment likelihood-change. To be precise, we start at the root as the ‘current node’ and then search the tree like below. During this search, we always keep track of the best attachment likelihood seen thus far, which we call dL_{opt} .

1. Calculate the attachment likelihood for the current node dL_{curr} . If $dL_{\text{curr}} > dL_{\text{opt}}$, then we update this value.
2. Calculate the attachment likelihood for all neighbors of the current node, dL_i .
3. Continue the search at step 1 for all neighbors that are within a tolerance Δ of the best attachment likelihood seen so far, i.e., whenever $dL_i > dL_{\text{opt}} - \Delta$.

Following this procedure, we search the tree, starting at the root, by following all paths that maintain a high enough attachment-likelihood, as defined by the tolerance, Δ , which can be set by the user.

As a third strategy, we can start this same search from several starting points to ensure that our tree-search does not get stuck in local minima. To be precise, we use our minimal-distance-based clustering described in the Methods-section, to find $\log(n)$ cluster-centers spread over the tree, where n is the number of nodes in the tree. We then follow the search described above from each of these cluster-centers, and only use the best attachment node that is found.

SI.B.4.2 Attaching the new cell to an edge

In the above, we have only calculated the loglikelihood change when the new cell was attached directly to a node, but it is of course possible that the likelihood is maximized when the cell is attached to an edge. Indeed, by attaching the cell to its optimal attachment node, we create a polytomy at this node, since it already had at least three connecting nodes. We therefore always follow a cell-attachment by resolving this newly created polytomy according to the general method that we will describe in Section SI.C.3.

SI.C Finding the tree with maximum posterior

In Section SI.B, we have derived an expression for the likelihood of the data *given a single tree*. However, we are interested in finding the tree with maximal posterior probability. For this, we should first choose a prior distribution over tree topologies and branch lengths, which we will just take to be uniform. Therefore, we get

$$P(T, \mathbf{t}|D) \propto P(D|T, \mathbf{t}). \quad (\text{S28})$$

Finding the most likely tree thus boils down to finding a tree that maximizes the likelihood of the data. We have already mentioned that we can maximize the branch lengths for a given topology using the gradient derived above, so the problem remains to find the best tree topology. However, the number of possible tree topologies is extremely large, and it is computationally infeasible to do a brute-force search for the maximal tree. Therefore, we will use the following scheme which is based on tree space search strategies learned from phylogenetics.

1. Start from a star-tree
2. Iteratively add the most likely internal node
3. Further resolve polytomies
4. Optimize branch times
5. Perform Subtree Prune and Regraft (SPR)-moves
6. Interchange nearest neighbors
7. Optimize branch times a final time

On the following pages, I will explain the separate steps.

SI.C.1 Start from a star-tree

To start, we create a leaf-node for each cell, and connect these leaves to a single root-node. We thus have a tree with $|C| + 1$ nodes, and $|C|$ edges. For this tree, we can calculate the loglikelihood and its gradient with respect to the branch lengths, which we use to do a first optimization of these branch lengths.

SI.C.2 Iteratively add the most likely internal node

Starting from the star-tree, we use a tree space search strategy loosely based on the popular ‘Neighbor-Joining’ heuristic [23]. For a pair of leaves in the tree, we can propose to add an ancestor between the root and the leaves (see Figure S38B). The new tree would then have $|C| - 2$ leaves connected to the root (r), while the other 2 leaves are connected to an ancestor a , which is in turn connected to the root. We here search for *the pair of leaves for which adding an ancestor increases the likelihood most*. We will describe below (Section SI.C.2.1) how to calculate this increase in tree likelihood.

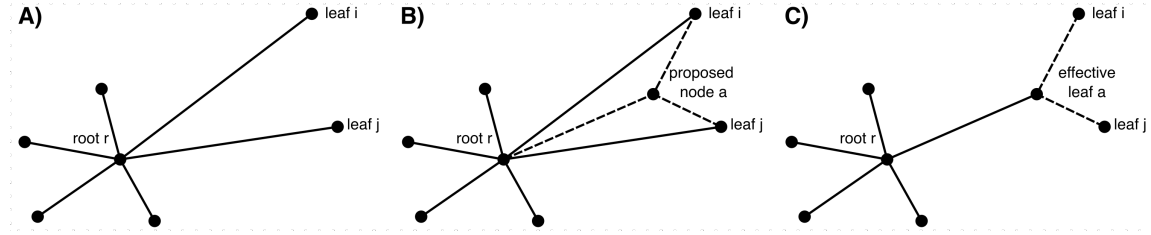


Figure S38. The iterative procedure of adding internal nodes to a star-tree. **A)** The iterative procedure starts from a star-tree in which all leaves are connected to a root-node. **B)** We can calculate the increase in tree likelihood that would result from adding an internal node to any pair of leafs. Here, we would add node a upstream of leafs i and j . Once we pick the optimal internal node to add, we can summarize the likelihood of the subtree downstream of a in the properties of an effective leaf a . Viewed from the root, we thus have a star-tree again, such that we can start again from **A**).

After this ancestor is added, we can summarize the newly created subtree in an effective leaf for the ancestor a , as described in Section SI.B.2.2 (see Figure S38C)). In this way, we have created a star-tree again and we can again search for the best pair of (effective) leaves to merge into a new ancestor. This is repeated until either the root has only 3 children left, or when no pair can be found that increases the tree likelihood.

SI.C.2.1 Finding the ancestor that increases the likelihood most

We want to calculate the likelihood increase of merging a pair of leaves k, l into an internal node a that connects to the root r . We thus want to calculate the likelihood for the tree T_{merge} where this ancestor is added, and compare it to the likelihood of the tree T_{orig} without the added ancestor. In the following, we will again use that we can summarize subtrees in effective leaves (Section SI.B.2.2), and we will denote all leaves other than k, l as a set R .

The likelihood for T_{orig} . It is convenient to summarize the subtree (T_R) comprising the root and all leaves other than k, l into an effective leaf. We get a star-tree with three leaves:

$$P(D|T_{\text{orig}}, \mathbf{t}) = P(D|T_R, \mathbf{t}) \\ \times \int dx_r \left[\frac{\sqrt{w_{r \rightarrow k}^* w_{r \rightarrow l}^* \bar{w}_r^R}}{(2\pi)^{3/2}} \exp \left(-\frac{1}{2} (w_{r \rightarrow k}^* (x_r - \mu_k)^2 + w_{r \rightarrow l}^* (x_r - \mu_l)^2 + \bar{w}_r^R (x_r - \bar{\mu}_r^R)^2) \right) \right],$$

where $\bar{\mu}_r^R = \frac{\sum_{i \in R} w_i^* \mu_i}{\bar{w}_r^R}$ and $\bar{w}_r^R = \sum_{i \in R} w_i^*$. We also need to be explicit about the time-dependence of the precisions, we have $w_{r \rightarrow k}^* = \frac{1}{t_{r \rightarrow k} + 1/w_k}$, and similar for $w_{r \rightarrow l}^*$. We have here written $t_{r \rightarrow k}$ to distinguish it from the branch length $t_{a \rightarrow k}$ that we will introduce later. Using the expression for the star-tree loglikelihood we get

$$L(D|T_{\text{orig}}, \mathbf{t}) = L(D|T_R, \mathbf{t}) + \frac{1}{2} \left[\log(w_{r \rightarrow k}^* w_{r \rightarrow l}^* \bar{w}_r^R) - \log(w_{r \rightarrow k}^* + w_{r \rightarrow l}^* + \bar{w}_r^R) \right. \\ \left. - w_{r \rightarrow k}^* (\bar{\mu}_r - \mu_k)^2 - w_{r \rightarrow l}^* (\bar{\mu}_r - \mu_l)^2 - \bar{w}_r^R (\bar{\mu}_r - \bar{\mu}_r^R)^2 \right] \quad (\text{S29})$$

The likelihood for T_{merge} . When we do merge the leaves k, l into an ancestor a , it is best to designate the new ancestor as the new root. The likelihood then takes a very similar shape, although the term $\bar{w}_r^R$ is replaced by $w_{a \rightarrow r}^* = \frac{1}{t_a + 1/\bar{w}_r^R}$ to account for the added diffusion time from the root to the ancestor:

$$P(D|T_{\text{merge}}, \mathbf{t}) = P(D|T_R, \mathbf{t}) \\ \times \int dx_a \left[\frac{\sqrt{w_{a \rightarrow k}^* w_{a \rightarrow l}^* w_{a \rightarrow r}^*}}{(2\pi)^{3/2}} \exp \left(-\frac{1}{2} (w_{a \rightarrow k}^* (x_a - \mu_k)^2 + w_{a \rightarrow l}^* (x_a - \mu_l)^2 + w_{a \rightarrow r}^* (x_a - \bar{\mu}_r^R)^2) \right) \right].$$

Performing the integral gives

$$L(D|T_{\text{merge}}, \mathbf{t}) = L(D|T_R, \mathbf{t}) + \frac{1}{2} \left[\log(w_{a \rightarrow k}^* w_{a \rightarrow l}^* w_{a \rightarrow r}^*) - \log(w_{a \rightarrow k}^* + w_{a \rightarrow l}^* + w_{a \rightarrow r}^*) \right. \\ \left. - w_{a \rightarrow k}^* (\bar{\mu}_a - \mu_k)^2 - w_{a \rightarrow l}^* (\bar{\mu}_a - \mu_l)^2 - w_{a \rightarrow r}^* (\bar{\mu}_a - \bar{\mu}_r^R)^2 \right] \quad (\text{S30})$$

The increase in loglikelihood. Combining Equations (S29) and (S30), we can calculate the increase in loglikelihood caused by merging leaves k, l into an ancestor:

$$\Delta L_{k,l} = L(D|T_{\text{merge}}, \mathbf{t}) - L(D|T_{\text{orig}}, \mathbf{t}) \\ = \frac{1}{2} \sum_g \left[\log(w_{a \rightarrow k}^* w_{a \rightarrow l}^* w_{a \rightarrow r}^*) - \log(w_{a \rightarrow k}^* + w_{a \rightarrow l}^* + w_{a \rightarrow r}^*) \right. \\ \left. - \log(w_{r \rightarrow k}^* w_{r \rightarrow l}^* \bar{w}_r^R) + \log(w_{r \rightarrow k}^* + w_{r \rightarrow l}^* + \bar{w}_r^R) \right. \\ \left. - w_{a \rightarrow k}^* (\bar{\mu}_a - \mu_k)^2 - w_{a \rightarrow l}^* (\bar{\mu}_a - \mu_l)^2 - w_{a \rightarrow r}^* (\bar{\mu}_a - \bar{\mu}_r^R)^2 \right. \\ \left. + w_{r \rightarrow k}^* (\bar{\mu}_r - \mu_k)^2 + w_{r \rightarrow l}^* (\bar{\mu}_r - \mu_l)^2 + \bar{w}_r^R (\bar{\mu}_r - \bar{\mu}_r^R)^2 \right], \quad (\text{S31})$$

where we thus have the following differences between the two tree topologies:

$$w_{a \rightarrow k}^* = \frac{1}{t_{a \rightarrow k} + \sigma_k^2} \rightarrow w_{r \rightarrow k}^* = \frac{1}{t_{r \rightarrow k} + \sigma_k^2} \\ w_{a \rightarrow r}^* = \frac{1}{t_a + 1/\bar{w}_r^R} \rightarrow \bar{w}_r^R \\ \bar{\mu}_a = \frac{w_{a \rightarrow k}^* \mu_k + w_{a \rightarrow l}^* \mu_l + w_{a \rightarrow r}^* \bar{\mu}_r^R}{w_{a \rightarrow k}^* + w_{a \rightarrow l}^* + w_{a \rightarrow r}^*} \rightarrow \bar{\mu}_r = \frac{w_{r \rightarrow k}^* \mu_k + w_{r \rightarrow l}^* \mu_l + \bar{w}_r^R \bar{\mu}_r^R}{w_{r \rightarrow k}^* + w_{r \rightarrow l}^* + \bar{w}_r^R} \quad (\text{S32})$$

Optimizing the branch lengths after the merge. Adding an internal node to the tree could in principle change the optimal branch lengths across the whole tree. However, performing a tree-wide branch length optimization after every proposed merge would quickly become computationally infeasible. Therefore, we assume that the optimal branch lengths are well-approximated by only optimizing the branch lengths in the subtree that is changed, i.e., we only optimize $t_{a \rightarrow k}, t_{a \rightarrow l}, t_{a \rightarrow r}$ for calculating $\Delta L_{k,l}$.

In addition, since we will iteratively add more and more ancestors to the tree, there is a large chance that the branch length $t_{a \rightarrow r}$ will have to be re-optimized later, either because another ancestor (a') is added upstream of a so that we optimize a branch length $t_{a' \rightarrow a}$, or because $t_{a \rightarrow r}$ simply becomes sub-optimal because the root changes position when merges are done in different parts of the tree. The same does not hold for the connection between the merged leaves k and l . As a result, it turns out that our tree-reconstruction yields the highest likelihood trees if we do the following time-optimization scheme:

1. First we optimize the sum $t_{a \rightarrow k} + t_{a \rightarrow l}$ by assuming that the ancestor a is not attached to the root. This will ensure that the sum of these times (which will thus not change in the following steps in the iteration) is optimized to capture the distance between the leaves k and l .
2. Then optimize all three times $t_{a \rightarrow k}, t_{a \rightarrow l}, t_{a \rightarrow r}$ while constraining the above sum.

SI.C.3 Further resolve polytomies

In the previous step we have iteratively merged pairs of child-nodes of the root into new ancestors, until no pair could further increase the tree likelihood. It turns out, however, that this procedure can result in downstream nodes that have more than 2 child-nodes, so-called polytomies. This for example happens when first an ancestor a is added upstream of leaves k and l , and then another ancestor (a') is added upstream of a and a leaf m , but the optimal times are such that $t_{a' \rightarrow a} = 0$. Of course, an edge with zero length doesn't change the likelihood of the tree, so we can remove the node a and directly connect a' to k, l, m . Since this tree results from an optimization, this configuration is indeed the best given the root-position at that point in the optimization. However, later in the iterative optimization, the root-position may have changed enough to warrant reconsideration of the topology $a \rightarrow k, l, m$. This turns out to be often the case. Therefore, we add a step in the procedure where we go over all nodes that still have more than 2 child-nodes, and we greedily add the ancestors downstream of these nodes that are mostly increasing the likelihood. Note that the necessary calculations are exactly the same as in the original merging procedure (Section SI.C.2), except that now we don't only consider the children of the root for merging, but also the children in other polytomies.

SI.C.4 Optimize branch lengths

Since in the iterative procedures above, we only maximized the branch lengths directly connected to the new ancestor, we may have drifted away from the global optimum for all branch lengths. Therefore, we do another round of gradient-based optimization of all branch lengths (see Section SI.B.3).

SI.C.5 Subtree pruning and regrafting

In phylogenetic tree reconstruction, tree moves are often used to search the tree space. These moves change the tree topology after which a new tree likelihood is calculated. An often-used tree move is the Subtree Prune and Regraft move. In this move, we pick a node and detach it from the tree, thereby also removing the subtree that hangs downstream from the node (Subtree Pruning). Then, this subtree is attached somewhere else in the tree (Regrafting).

SI.C.5.1 Pruning a subtree

In *Bonsai*, we have implemented two strategies for picking the subtree that is pruned. First, we can pick a random node and detach its downstream subtree. Second, we order the nodes by the length of their upstream branch, and then sequentially try to prune and regraft the subtrees downstream of these nodes. This strategy is based on our finding that nodes that are connected through a long upstream branch are typically least sure about their position on the tree. Indeed, the long branch signals that the gene expression is dissimilar between the two nodes, and there may thus be a position where this subtree fits better. In the current *Bonsai*-implementation, we use the second strategy by default.

SI.C.5.2 Regrafting the subtree

For re-attaching the subtree, we summarize the subtree as an effective leaf as described in SI.B.2.2, and then follow the strategy outlined in SI.B.4 to optimally place this subtree back on the resulting tree.

SI.C.6 Interchange nearest neighbors

Another well-known tree move is a so-called Nearest Neighbor Interchange (NNI). In this move, we pick an arbitrary edge in the tree that is not connected to a leaf, let's say the edge connects nodes k and l . We then consider the sub-trees connected to the nodes k and l . In a fully binary tree, i.e., a tree without any polytomies, we would always have 4 sub-trees. Now an NNI-move consists of reconnecting these 4 sub-trees in different ways. For example, if subtrees T_A, T_B were downstream of k and T_C, T_D downstream of l , an NNI-move could connect T_A, T_C to one node and T_B, T_D to another. The nearest-neighbors of these sub-trees have thus been interchanged.

The trees that are reconstructed by *Bonsai* often have polytomies, so we need a way to generalize this NNI-move to having more than 4 sub-trees. We chose to do this by doing the following:

1. We pick an arbitrary edge that connects two internal nodes, say k and l .
2. We remove the edge and node k by taking all sub-trees that were attached to node k and attach them to node l .
3. We now consider the node l as the root of a star-tree. We can now apply the same procedure of iteratively adding ancestors as described in SI.C.2.

It can be seen that this is a generalization of the NNI-move in the case of 4 sub-trees. In that case, we can maximally add one ancestor, and all possibilities are to group the subtrees as $(T_A, T_B), (T_C, T_D), (T_A, T_C), (T_B, T_D)$ or rather $(T_A, T_D), (T_B, T_C)$.

Randomly interchanging nearest neighbors. To make sure we are not getting stuck in a local optimum, we do several rounds of random NNI-moves. We follow the steps as outlined above, but now we randomly pick a pair of child-nodes for which we add an ancestor. This random choice is weighted by the tree likelihoods, such that the probability of merging q, r is $p_{q,r} = \frac{P(D|T_{\text{merge } q,r}, \mathbf{t})}{\sum_{i,j \in C_l(l)} P(D|T_{\text{merge } i,j}, \mathbf{t})}$.

Greedily interchanging nearest neighbors. We end by performing NNI-moves in a greedy fashion. That is, we determine the loglikelihood increase that an NNI-move would bring for every edge in the tree and perform the NNI-move that maximizes this increase. This is repeated until no NNI-move significantly increases the likelihood.

SI.C.7 Optimize branch lengths a final time

Again, in the NNI-moves, only the connected branch lengths are optimized. We therefore do another round of branch length optimization to end with the tree that is most likely to describe the data.

SI.D Speeding up the tree reconstruction

Current single-cell datasets can have over 10 million cells, a number that can even complicate running a method that scales quadratically in this number. The above described *Bonsai* implementation needs to compare the loglikelihood increase for merging all pairs of cells at every addition of a new internal node. Since the number of internal nodes that will be added is also proportional to $|C|$, and because the calculation of one loglikelihood increase scales with the number of genes $|G|$, the computational scaling is $O(|C|^3|G|)$. This is clearly problematic. In this Section we show two computational tricks that speed *Bonsai* up to a manageable scaling.

SI.D.1 Calculating upper bounds for merging likelihoods given a change in the root's position

In the tree reconstruction procedure, we repeatedly have to calculate the likelihood increase upon merging pairs of leaves into an ancestor. In every iteration, we calculate the likelihood increase for merging many possible pairs, but only one of these pairs leads to the largest likelihood increase, say pair m, n . We will thus only merge the maximal pair m, n into a new ancestor. Because this merge will influence the position of the root, Equation (S31) shows that this will also affect the loglikelihood increase due to merging all other pairs of leaves. In principle, all calculations should thus be repeated.

However, Equation (S31) also shows that adding an ancestor somewhere else will *only* affect the likelihood increase for pair (k, l) through the effective position and precision of the root when accounting for all other leaves, i.e., through $\bar{\mu}_r^R$, and $\bar{w}_r^R$. In turn, these depend on the effective position and precision when accounting for all leaves via $\bar{\mu}_r^R = \frac{\bar{\mu}_r \bar{w}_r - w_k^* \mu_k - w_l^* \mu_l}{\bar{w}_r - w_k^* - w_l^*}$ and $\bar{w}_r^R = \bar{w}_r - w_k^* - w_l^*$. We can therefore try to estimate an upper bound on what $\Delta L_{k,l}$ can maximally become given a certain amount of movement in the root position. For this, we will first compute a linear approximation of $\Delta L_{k,l}$ as a function of $(\bar{\mu}_r, \bar{w}_r)$. Then, we will take an ellipsoid-region around the current $(\bar{\mu}_r, \bar{w}_r)$ and calculate the maximum for the linear approximation in this region. This, we will use as an upper bound for $\Delta L_{k,l}$ as long as the effective root position does not leave the ellipsoid ball. When it does leave the ellipsoid ball, we should recalculate the upper bounds. Note that this is

not a strict mathematical upper bound, because the values of $\Delta L_{k,l}$ could be underestimated by the linear approximation, but this underestimation will be very small because for small ellipsoid regions this linear approximation will be very good. Even stronger, any underestimation will be completely offset by the much larger overestimation we do by taking the maximum in the ellipsoid. Indeed, for calculating the maximum we will assume (see below) that the root position and precision travelled to exactly the right point at the border of a very high-dimensional ellipsoid. Since this will happen only with probability zero, we will in all real cases overestimate the possible $\Delta L_{k,l}$.

A linear approximation of $\Delta L_{k,l}$. To get the derivative of $\Delta L_{k,l}$ with respect to $\bar{\mu}_r$ and $\bar{w}_r$, it is convenient to re-write the expression for $\Delta L_{k,l}$ using the following identity:

$$\sum_i w_i \left(\mu_i - \frac{\sum_k w_k \mu_k}{\sum_j w_j} \right)^2 = \frac{1}{\sum_i w_i} \sum_{i,j,i < j} w_i w_j (\mu_i - \mu_j)^2. \quad (\text{S33})$$

Using this, we get

$$\begin{aligned} \Delta L_{k,l} = \frac{1}{2} & \left[\log(w_{a \rightarrow k}^* w_{a \rightarrow l}^* w_{a \rightarrow r}^*) - \log(w_{a \rightarrow k}^* + w_{a \rightarrow l}^* + w_{a \rightarrow r}^*) \right. \\ & - \log(w_{r \rightarrow k}^* w_{r \rightarrow l}^* \bar{w}_r^R) + \log(w_{r \rightarrow k}^* + w_{r \rightarrow l}^* + \bar{w}_r^R) \\ & - \frac{1}{\bar{w}_a} (w_{a \rightarrow k}^* w_{a \rightarrow l}^* (\mu_k - \mu_l)^2 + w_{a \rightarrow k}^* w_{a \rightarrow r}^* (\mu_k - \bar{\mu}_r^R)^2 + w_{a \rightarrow l}^* w_{a \rightarrow r}^* (\mu_l - \bar{\mu}_r^R)^2) \\ & \left. + \frac{1}{\bar{w}_r} (w_{r \rightarrow k}^* w_{r \rightarrow l}^* (\mu_k - \mu_l)^2 + w_{r \rightarrow k}^* \bar{w}_r^R (\mu_k - \bar{\mu}_r^R)^2 + w_{r \rightarrow l}^* \bar{w}_r^R (\mu_l - \bar{\mu}_r^R)^2) \right]. \end{aligned} \quad (\text{S34})$$

The derivative with respect to $\bar{\mu}_r$ is given by

$$\frac{\partial \Delta L_{k,l}}{\partial \bar{\mu}_r} = \frac{\bar{w}_r}{\bar{w}_r^R} \left(\left(\frac{w_{a \rightarrow k}^* w_{a \rightarrow r}^*}{\bar{w}_a} - \frac{w_{r \rightarrow k}^* \bar{w}_r^R}{\bar{w}_r} \right) (\mu_k - \bar{\mu}_r^R) + \left(\frac{w_{a \rightarrow l}^* w_{a \rightarrow r}^*}{\bar{w}_a} - \frac{w_{r \rightarrow l}^* \bar{w}_r^R}{\bar{w}_r} \right) (\mu_l - \bar{\mu}_r^R) \right), \quad (\text{S35})$$

and with respect to $\bar{w}_r$ by

$$\begin{aligned} \frac{\partial \Delta L_{k,l}}{\partial \bar{w}_r} = \frac{1}{(\bar{w}_r^R)^2} & \left((w_{a \rightarrow r}^* - \bar{w}_r^R - \frac{(w_{a \rightarrow r}^*)^2}{\bar{w}_a} + \frac{(\bar{w}_r^R)^2}{\bar{w}_r}) \right. \\ & + \frac{(\mu_k - \bar{\mu}_r^R)^2}{(\bar{w}_r^R)^2} \left(\frac{w_{r \rightarrow k}^* (\bar{w}_r^R)^2}{\bar{w}_r} - \frac{w_{a \rightarrow k}^* (w_{a \rightarrow r}^*)^2}{\bar{w}_a} \right) \\ & + \frac{(\mu_l - \bar{\mu}_r^R)^2}{(\bar{w}_r^R)^2} \left(\frac{w_{r \rightarrow l}^* (\bar{w}_r^R)^2}{\bar{w}_r} - \frac{w_{a \rightarrow l}^* (w_{a \rightarrow r}^*)^2}{\bar{w}_a} \right) \\ & + \frac{2(\bar{\mu}_r - \bar{\mu}_r^R)}{(\bar{w}_r^R)} \left(\frac{w_{a \rightarrow k}^* w_{a \rightarrow r}^*}{\bar{w}_a} (\mu_k - \bar{\mu}_r^R) + \frac{w_{a \rightarrow l}^* w_{a \rightarrow r}^*}{\bar{w}_a} (\mu_l - \bar{\mu}_r^R) - \frac{w_{r \rightarrow k}^* \bar{w}_r^R}{\bar{w}_r} (\mu_k - \bar{\mu}_r^R) - \frac{w_{r \rightarrow l}^* \bar{w}_r^R}{\bar{w}_r} (\mu_l - \bar{\mu}_r^R) \right) \\ & + \frac{(w_{a \rightarrow r}^*)^2}{(\bar{w}_a)^2 (\bar{w}_r^R)^2} (w_{a \rightarrow k}^* w_{a \rightarrow l}^* (\mu_k - \mu_l)^2 + w_{a \rightarrow k}^* w_{a \rightarrow r}^* (\mu_k - \bar{\mu}_r^R)^2 + w_{a \rightarrow l}^* w_{a \rightarrow r}^* (\mu_l - \bar{\mu}_r^R)^2) \\ & \left. - \frac{1}{(\bar{w}_r)^2} (w_{r \rightarrow k}^* w_{r \rightarrow l}^* (\mu_k - \mu_l)^2 + w_{r \rightarrow k}^* \bar{w}_r^R (\mu_k - \bar{\mu}_r^R)^2 + w_{r \rightarrow l}^* \bar{w}_r^R (\mu_l - \bar{\mu}_r^R)^2) \right) \end{aligned} \quad (\text{S36})$$

Let us now denote the original position and precision of the root by the usual $\bar{\mu}_{rg}$ and $\bar{w}_{rg}$, and the movement in these variables by ξ_g^μ and ξ_g^w . With the above derivatives we then get the linear approximation:

$$\Delta L_{k,l}(\bar{\mu}_{gr} + \xi_g^\mu, \bar{w}_{gr} + \xi_g^w) - \Delta L_{k,l}(\bar{\mu}_{gr}, \bar{w}_{gr}) \approx \sum_g \frac{\partial \Delta L_{k,l}}{\partial \bar{\mu}_{gr}} \xi_g^\mu + \sum_g \frac{\partial \Delta L_{k,l}}{\partial \bar{w}_{gr}} \xi_g^w \quad (\text{S37})$$

Choosing ellipsoid regions around the root position and precision. The above linear approximation allows us to estimate the increase in loglikelihood given a certain root movement (ξ_g^μ, ξ_g^w) . We now want to constrain the allowed movements to a certain ellipsoid around $(\bar{\mu}_{gr}, \bar{w}_{gr})$ to be able to estimate an upper bound. It is important to note that the validity of the upper bound will not depend on the choice of the ellipsoid. This choice will however determine for how many steps the upper bound will remain valid. We thus want to pick the ellipsoid such that the root position and precision remain within the ellipsoid for as many steps as possible, while still allowing for a tight upper bound.

It is a reasonable choice to assume that the change in root position upon merging two leaves into an ancestral node scales with the standard deviation in the current position of the root and the current number of children of the root (n_c), i.e.:

$$\xi_g^\mu \propto \frac{1}{\sqrt{n_c \bar{w}_{gr}}}. \quad (\text{S38})$$

The reasoning behind this is that x_{gr} is the weighted average of the positions of n_c positions, and we know (when subtracting the mean) that $x_{gr} \sim N(0, \sigma^2 = \frac{1}{W(r)^g})$. If we make the approximation that the weights are all the same and that the original positions are drawn from some Gaussian, we get that these individual positions must be distributed as $x_{gi} \sim N(0, \sigma^2 = \frac{n_c}{W(r)^g})$. Now, we are taking two of these x_{gi} out of the average and we add more or less the average of the two. We want to know the distribution of the change:

$$\xi_g^\mu = \frac{\sum_i x_{gi}}{n_c} - \frac{\sum_i x_{gi} - \frac{1}{2}(x_{g1} - x_{g2})}{n_c - 1} \approx \frac{\frac{1}{2}(x_{g1} + x_{g2})}{n_c} \sim N\left(0, \frac{1}{4n_c^2} \frac{2n_c}{\bar{w}_{gr}}\right) \sim N\left(0, \frac{1}{2n_c \bar{w}_{gr}}\right).$$

For a typical change in root position, we therefore expect ξ^μ to scale according to (S38). Therefore, we pick the dimensions of the ellipsoid to follow the same scaling.

Similarly, for the change in root precision, a good assumption is that it will scale as

$$\xi_g^w \propto \frac{\bar{w}_{gr}}{n_c}, \quad (\text{S39})$$

because we know $\bar{w}_{gr} = \sum_i w_i^*$. Removing two of these w_i^* and replacing it by one precision for the ancestor, we can approximate $\xi_g^w \propto \frac{\bar{w}_{gr}}{n_c}$.

In addition, we would like the root to remain in the ellipsoid for a certain number of merges (n_{steps}). We here assume that the root position follows a random walk in each gene-dimension with steps in both the positive and negative direction. This would lead to a scaling of $\xi_g^\mu \propto \sqrt{n_{\text{steps}}}$. In contrast, the root-precision will only decrease when child-nodes are merged, so there we expect

$\xi_g^w \propto n_{\text{steps}}$. We thus get

$$\xi_g^\mu \propto \sqrt{\frac{n_{\text{steps}}}{n_c \bar{w}_{gr}}} \quad \text{and} \quad \xi_g^w \propto \frac{n_{\text{steps}} \bar{w}_{gr}}{n_c} \quad (\text{S40})$$

Taking this into account, we choose ellipsoids for the position and precision as

$$\begin{aligned} E_\mu &:= \left\{ \xi_g^\mu \mid \sum_g \frac{n_c \bar{w}_{gr}}{n_{\text{steps}}} (\xi_g^\mu)^2 < 1 \right\}, \\ E_w &:= \left\{ \xi_g^w \mid \sum_g \frac{n_c^2}{\bar{w}_{gr}^2 n_{\text{steps}}^2} (\xi_g^w)^2 < 1 \right\}. \end{aligned} \quad (\text{S41})$$

Note that with the n_{steps} parameter we can tune the size of the ellipse, and with that, the trade-off between for how many steps the upper bound can be used and how tight the upper bounds are.

The final expression for the upper bound. Let us now introduce new variables

$$(\eta_g^\mu, \eta_g^w) = \left(\sqrt{\frac{n_c \bar{w}_{gr}}{n_{\text{steps}}}} \xi_g^\mu, \frac{n_c}{\bar{w}_{gr} n_{\text{steps}}} \xi_g^w \right). \quad (\text{S42})$$

We then get

$$E_\mu := \left\{ \eta_g^\mu \mid \sum_g (\eta_g^\mu)^2 < 1 \right\} \quad \text{and} \quad E_w := \left\{ \eta_g^w \mid \sum_g (\eta_g^w)^2 < 1 \right\} \quad (\text{S43})$$

For any $(\xi_g^\mu, \xi_g^w) \in E_\mu \times E_w$ we get that the linear approximation becomes

$$\begin{aligned} &\Delta L_{k,l}(\bar{\mu}_{gr} + \xi_g^\mu, \bar{w}_{gr} + \xi_g^w) - \Delta L_{k,l}(\bar{\mu}_{gr}, \bar{w}_{gr}) \\ &\approx \frac{\partial \Delta L_{k,l}}{\partial \bar{\mu}_{gr}} \xi_g^\mu + \frac{\partial \Delta L_{k,l}}{\partial \bar{w}_{gr}} \xi_g^w \\ &= \sum_g \frac{\partial \Delta L_{k,l}}{\partial \bar{\mu}_{gr}} \sqrt{\frac{n_{\text{steps}}}{n_c \bar{w}_{gr}}} \eta_g^\mu + \sum_g \frac{\partial \Delta L_{k,l}}{\partial \bar{w}_{gr}} \frac{\bar{w}_{gr} n_{\text{steps}}}{n_c} \eta_g^w \end{aligned} \quad (\text{S44})$$

We can now recognize this as a dot product when we define the vectors $\nu_g^\mu = \frac{\partial \Delta L_{k,l}}{\partial \bar{\mu}_{gr}} \sqrt{\frac{n_{\text{steps}}}{n_c \bar{w}_{gr}}}$ and $\nu_g^w = \frac{\partial \Delta L_{k,l}}{\partial \bar{w}_{gr}} \frac{\bar{w}_{gr} n_{\text{steps}}}{n_c}$. This means we can get an upper bound for the possible loglikelihood increase in the ellipsoid by picking η_g^μ of length 1 and in the direction of ν_g^μ , and similarly for η_g^w . This yields

$$\Delta L_{k,l}(\bar{\mu}_{gr} + \xi_g^\mu, \bar{w}_{gr} + \xi_g^w) - \Delta L_{k,l}(\bar{\mu}_{gr}, \bar{w}_{gr}) < \left| \frac{\partial \Delta L_{k,l}}{\partial \bar{\mu}_{gr}} \sqrt{\frac{n_{\text{steps}}}{n_c \bar{w}_{gr}}} \right| + \left| \frac{\partial \Delta L_{k,l}}{\partial \bar{w}_{gr}} \frac{\bar{w}_{gr} n_{\text{steps}}}{n_c} \right| := \Delta L_{k,l}^{\text{UB}} \quad (\text{S45})$$

Using the upper bound information. To use the upper bound information to speed up the tree reconstruction, we will follow the following steps:

1. We will in the first round of adding an ancestor calculate the maximal loglikelihood increase for all pairs $\Delta L_{k,l}$, and a corresponding upper bound $\Delta L_{k,l}^{\text{UB}}$ for a given ellipsoid size set by n_{steps} .
2. To prepare for the next round, we make a list of all pairs, sorted in decreasing order for their upper bound. These upper bounds were thus calculated for the original root-position and -precision: $(\bar{\mu}_r^0, \bar{w}_r^0)$.
3. In the next round we calculate the true loglikelihood increase given the new root position and precision: $(\bar{\mu}_r^1, \bar{w}_r^1)$ for the first pair in the list. For this pair, we thus estimated the largest upper bound for its loglikelihood increase. We compare its true loglikelihood increase with the upper bound for the second pair in the list. If the loglikelihood increase for the first pair is larger than the upper bound of the second pair, we know that we have found the best pair to merge. We can add the corresponding internal node and move on to the next round. If not, we proceed to calculating the loglikelihood increase for the second pair, and so forth until we have found the best pair.
4. After each addition of an internal node, we need to re-calculate the root position and precision. At that point, we test if these position and precision still fall within the ellipsoid region for which we estimated the upper bounds. If not, we need to calculate new upper bounds in the next round. In addition, for each new internal node, we need to calculate the loglikelihood increase of merging this node to all existing child-nodes. We then also calculate upper bounds for these new pairs and add them to the sorted list.

SI.D.1.1 Dynamically setting the ellipsoid size

It is hard to determine the optimal setting of the ellipsoid size (as controlled by the parameter n_{steps}). Therefore, in the *Bonsai* implementation, we monitor for every round how many pairs we needed to check from the sorted list of upper bounds. If this becomes too large, we decrease the ellipsoid size, as the upper bounds are not sufficiently tight. If the number of checked pairs is small, we increase the ellipsoid size, because we can probably get away with performing fewer calculations of new upper bounds.

SI.D.2 Limiting the number of candidate-pairs

Although the above described usage of upper bound estimates make sure we do not have to compute $|C|^2$ increases in likelihood at every round of adding an internal node, it is still necessary to perform this computation several times, i.e., every time that the root has moved outside of the ellipsoid. It is therefore necessary to further reduce *Bonsai*'s computational footprint to run the largest datasets.

We will here use that, for every round of adding an internal node, we only need to find *the best* pair of nodes to merge into an ancestor, i.e., we don't need a full ranking of all pairs of nodes. We therefore suggest to limit the search for the best pair by calculating the k nearest neighbors for every node based on a simple Euclidean distance in LTQ-space. We have seen in Equation (S34) that the precise loglikelihood increase for merging a certain pair of nodes is a complicated function of the positions and the uncertainties on these positions for the two nodes and the root. However, it

is clear that nodes that are close together and far from the root are likely candidates for having a high loglikelihood increase. Correspondingly, a node-pair that is far away from each other is never going to be the best pair to merge in the next round.

In practice, before the first round we calculate the k nearest neighbors based on Euclidean distance for every child-node of the root, where k -values as low as $5 - 10$ have been found to give the optimal tree-reconstruction on simulated datasets. Then we calculate loglikelihood increases only between each node and its nearest neighbors. The best pair of nodes is connected to a new ancestor, and this new ancestor inherits the union of the nearest neighbors of its downstream nodes. The nearest-neighbor relations are re-computed after a certain number of new ancestors has been added (typically after $0.1|C|$ new ancestors).

It is important to stress that we use the nearest-neighbors only to find the one pair of nodes that is most beneficial to merge, and for that we restrict our search to $|C|k$ candidates. This is a safe restriction to make, even though it is important to acknowledge that it is generally very hard to capture the full structure of a dataset by calculating a nearest-neighbor graph, and that it is even hard to correctly call nearest neighbors given the typical noisyness of single cell data. In other words, the likelihood that the one best pair in the whole dataset is among the many nearest-neighbors that we calculate is still very close to 1.

SI.D.3 The effect on *Bonsai*'s computation time

These computational advancements drastically reduce the computational footprint of the *Bonsai*-tree reconstruction. In Figure 3 one can see that *Bonsai* scales sub-quadratically in the number of cells.

SI.E Creating simulated scRNA-seq datasets

We have tested the performance of *Bonsai* on various simulated datasets. In particular, in Figure 2, and Supplementary Figures S2-S15, we describe the results of *Bonsai* on simulated scRNA-seq data. In general, the creation of such simulated datasets consists of two steps: 1) generating ground truth values, and 2) adding realistic measurement noise to these ground truth values. As explained in Section SI.B.1.2, for scRNA-seq data, the true underlying physical variables are the Log Transcription Quotients (LTQs). Step 1) of simulating the datasets thus consists of generating an LTQ vector for each cell, while step 2) comprises simulating measured mRNA counts given these LTQ values. Whereas the first step depends on the simulator's choice of what is a reasonable way to mimic a true dataset, there is a principled way of performing the second step. We will therefore first describe the second step, which remains the same across the seven simulated scRNA-seq datasets, and then describe how the LTQs were generated for the different datasets. Note that the simulation scripts can also all be found in the GitHub-repository (<https://github.com/dhdegroot/Bonsai-data-representation> [24]) in the subfolder `paper_figure_scripts_and_notebooks/simulating_datasets`.

SI.E.1 Sampling mRNA-counts given LTQ-values

Remember that the LTQs are defined as

$$x_{gc} = \log \left(\frac{\langle m_{gc} \rangle}{\sum_{g'} \langle m_{g'c} \rangle} \right), \quad (\text{S46})$$

This means that $\exp(x_{gc}) = \frac{\langle m_{gc} \rangle}{\sum_{g'} \langle m_{g'c} \rangle}$. To get a realistic estimate of the sum of expected mRNA-counts that appears in the denominator, we chose to just sample the sum of mRNA-counts per cell from a real dataset, specifically a human pancreatic dataset by Baron et al. [25]. In this dataset, the mean total mRNA-count per cell is 5828, but it varies from 1201 to 34 509. Once we have picked a total count (N_c) for each cell and use it as a proxy for $\sum_{g'} \langle m_{g'c} \rangle$ we get $N_c \exp(x_{gc}) = \langle m_{gc} \rangle$, so that now the expected mRNA-number is fixed for each gene and each cell. As mentioned in Section SI.B, and in [3], and as experimentally proven in [16], the true counts are then Poisson samples with these given means. Thus, for every gene-cell pair, we sample a count, n_{gc} , from a Poisson distribution with mean $N_c \exp\{x_{gc}\}$. These counts n_{gc} are the data on which we can now test *Bonsai*, and the LTQs x_{gc} are stored as the corresponding ground truth information about the cells' gene expression states.

SI.E.2 Simulating LTQ-values for each cell

We chose to simulate 7 datasets, 6 of which are based on an underlying tree structure, while the other mimics a dataset with 7 strongly separated celltypes. The reason that we based six datasets on a tree structure is that trees provide a way to create non-trivial structure in high-dimensional datasets, while other sampling schemes often result in equidistant data points. The first simulated dataset is based on a binary tree with constant branch lengths, after which we generate more datasets by 1) sampling varying branch lengths, 2) creating an unbalanced tree, 3) enforcing a realistic lower-dimensional covariance structure, or combinations thereof. These simulations can all be reproduced using the script `paper_figure_scripts_and_notebooks/simulating_datasets/simulate_unbalanced_randomtimes_realcovariance.py` that can be found in the *Bonsai-data-representation* GitHub-repository [24]. The basic ingredients for the simulated datasets are:

- All simulated datasets will have 17 499 genes, and 1024 cells.
- For each gene, we pick a random gene from the Baron-dataset [25], and use its corresponding mean log-transcription quotient, here estimated as $\mu_g = \log \left(\frac{N_g}{\sum_{g'} N_{g'}} \right)$.
- In the extensive analysis done in [3], it was noted that the gene variances in the Baron dataset were roughly exponentially distributed with mean 2. For the simulated datasets, we therefore randomly sample a gene variance, v_g , from an exponential distribution with mean 2 for each gene.

SI.E.2.1 A binary tree with constant branch lengths

To simulate the generation of gene expression states along a binary tree with constant branch lengths, we take the following steps:

1. We assign the zero vector of length 17 499 as the root coordinates: $\mathbf{y}_{\text{root}} = \mathbf{0}$.
2. To generate the coordinates for the two children of the root, we sample a change in gene expression for each gene from a Gaussian distribution with variance tv_g , where the branch length t is always equal to 1 in this dataset. The two child-nodes of the root are thus given coordinates $\mathbf{y}_{\text{child0}} = \mathbf{y}_{\text{root}} + \boldsymbol{\delta}_0$, and $\mathbf{y}_{\text{child1}} = \mathbf{y}_{\text{root}} + \boldsymbol{\delta}_1$, where the gene expression changes δ_{0g} are drawn from a Gaussian distribution with variance v_g .

3. We repeat the previous step for 10 generations. For example, we generate coordinates for the child-node downstream of child 0 as $\mathbf{y}_{\text{child00}} = \mathbf{y}_0 + \boldsymbol{\delta}_{00}$. As such, we end up with 1024 cells in the last generation. For the simulated dataset, we only take the coordinates of the cells from this last generation.
4. Now that each cell in the dataset is assigned a coordinate vector, we ensure that the means and variances are realistic for gene expression data. For each gene, we center the data by subtracting the mean across cells, after which we rescale the coordinates such that the variance becomes the prescribed variance v_g , and add μ_g to match the prescribed mean.
5. Finally, the definition of log-transcription quotients demands that for any LTQ-vector, $\mathbf{x}$, we have $\sum_g \exp(x_g) = 1$. We thus subtract a constant for each cell's LTQ vector that ensures this.

Given the simulated LTQ-values per cell, we now sample the captured mRNA counts as outlined in Section SI.E.1.

SI.E.2.2 Varying branch lengths

For the dataset based on a tree with varying branch lengths, we follow the same steps as for the simple binary tree, except that the gene expression changes are not drawn from a Gaussian distribution with variance $1v_g$, but with a variance $t_i v_g$, where t_i is independently drawn for each new child-node. We draw these branch lengths between 0.5 and 2, and uniformly in log-scale, i.e., $\log(t_i) \in \text{Unif}[\log(0.5), \log(2)]$

SI.E.2.3 An unbalanced tree

To create a dataset based on an unbalanced tree, i.e., a tree with different numbers of leaves on different sides of the tree, we modify the scheme outlined in SI.E.2.1, because we will no longer add two children to every internal node for 10 generations. Instead, we create a list of leaf-nodes, which we initialize with only the root-node. Then, at each step, we take a random leaf-node from the list and make it an internal node by adding two child-nodes downstream of it. These two child-nodes then also replace their parent in the list of leaf-nodes. We repeat this 1023 times, such that 1024 leaves are created.

SI.E.2.4 A tree embedded in a lower-dimensional subspace

In two of the datasets, we mimic the covariance structure of a real scRNA-seq dataset. This is to address the possible criticism that the above-created datasets were created by an independent stochastic process in each gene, while in a real dataset, the gene expression changes may occur in a lower-dimensional subspace. This could create a non-trivial covariance structure between the genes, and we want to test with this dataset if *Bonsai* can still reconstruct a faithful representation despite that.

To find a realistic covariance structure between genes, we start by taking the Baron dataset again [25], and we infer the log-transcription quotients (LTQs) for this data using *Sanity* [3]. Since *Sanity* infers a Gaussian approximation of the posterior distribution of the LTQs, we get the most realistic LTQs by sampling from this Gaussian posterior. We therefore sample an LTQ-value for every gene-cell-pair from the posterior distribution, getting realistic LTQs in a matrix X_{gc} . We

center X_{gc} by subtracting the mean LTQ per gene to get $\bar{X}_{gc}$, and then perform a singular value decomposition to get matrices U_{gk}, σ_k, V_{kc} that satisfy for all gene-cell-pairs:

$$\bar{X}_{gc} = \sum_k U_{gk} \sigma_k V_{kc}^T. \quad (\text{S47})$$

We now want to create an LTQ-matrix $\bar{X}_{gc}^{\text{sim}}$ that results from a ground truth tree-structure, and that has a similar covariance structure as $\bar{X}_{gc}$. For this, we are using that the columns of U_{gk} contain the PCA-components in gene-space, and that $\frac{\sigma_k^2}{|C|}$ captures the variance along the k -th component. In the following, we will keep these two components the same. In contrast, the columns of V_{kc} contain the coordinates of the cells in the PCA-basis. We will simulate a matrix V_{kc}^T by modeling diffusion along a binary tree. This is thus the same as in Section S1.E.2.1, except that now the diffusion happens in the PCA-basis rather than in the gene-basis. Given the resulting cell-vectors in the PCA basis, we scale each component to have a variance of $\frac{\sigma_k^2}{|C|}$, after which we left-multiply with U_{gk} to rotate the simulated vectors back from the PCA-basis into the gene-basis. The resulting dataset, by construction, has the same covariance structure as the original real dataset.

SI.E.2.5 A pseudo-bulk-based dataset

To create a simulated dataset not based on a tree, we start by taking a single-cell dataset of the mouse cortex and hippocampus (Zeisel et al. [26]), of which the cells were annotated into 7 clusters. We create so-called pseudobulked counts by summing all mRNA counts for cells from the same celltype. We thus obtain 7 mRNA count vectors, one for each celltype, on which we run *Sanity* to obtain the corresponding ‘celltype LTQs’. Subsequently, we assign each of the 1024 simulated cells to one of the 7 celltypes in proportion to the abundance of the celltype in the original dataset. The final cell-type abundances range from 34 cells for the rarest to 320 cells for the most abundant celltype. We then create the LTQs for each cell by taking its celltype-LTQ vector and adding a contribution drawn from a Gaussian with variance 0.05. This small amount of intra-celltype variation is to be expected in biological datasets, and ensures that the kNN-test presented in S15 is well-defined. Finally, we again ensure that the LTQs satisfy the constraint $\sum_g \exp(x_g) = 1$, and we sample the mRNA counts as described in S1.E.1.

SI.F Creating more general simulated datasets

In Section S1.E, we describe how we simulated scRNA-seq datasets including the sampling of mRNA counts. However, we have also used three test datasets in which we directly simulated the input coordinates that were given to the various visualization methods that we test. We will here shortly outline how these data were simulated. The simulation scripts and the necessary commands to run them can also all be found by reading through the Jupyter Notebooks provided in the GitHub-repository (<https://github.com/dhdegroot/Bonsai-data-representation> [24]) in the subfolder `paper_figure_scripts_and_notebooks`.

SI.F.1 Noise-free datasets with increasing dimensionality

Using Figure S12, we argued that distances between objects can be captured better and better at increasing dimensionality. For this figure, we simulated a set of datasets with 100 objects in

either 2, 10, 100, 1000 or 10 000 dimensions. For each gene, we sample a gene-variance v_g from an exponential distribution with mean 2. Then, for each cell, we draw a cell-variance t_c uniform in log-scale between 0.1 and 10, i.e., $\log t_i \in \text{Unif}[\log(0.1), \log(10)]$, after which we rescale all the t_i such that $\frac{1}{100} \sum_{i=1}^{100} t_i = 1$. Finally, we just draw the coordinate for object i and gene g from a Gaussian distribution with mean zero and variance $t_i v_g$. These coordinates are stored as the input to *Bonsai*.

SI.F.2 Noisy datasets with an increasing number of objects

Through Figure S14, we showed that tree-structures can automatically use nearby cells to get more accurate distance estimates in noisy datasets. For this, we will use the 100 points in 100 dimensions from the simulation in SI.F.1 as cluster-centers; these will correspond to the *true positions*. Then, we mimic having noisy readouts from these cluster-centers by creating either 1, 2, 5, 10, or 20 objects around each center, thus creating in total 100, 200, 500, 1000 or 10 000 objects. The *measured coordinates* of these objects are given by the cluster-center-coordinates plus a noise term drawn from a Gaussian with variance 2.5. When *Bonsai* is used to reconstruct a tree and infer the distances, it is given the measured coordinates and the correct size of the measurement noise as input. The inferred object-to-object-distances are compared to the distances between the true positions, i.e., between the cluster-center-coordinates.

To make the task of inferring distances between objects perfectly comparable across the datasets with increasing numbers of objects, we use random seeds. In this way, we make sure that the first object around each cluster-center always has the exact same coordinates in each dataset, that the second object around each cluster-center has the same coordinates in the last 4 datasets, and so on. The distance-comparison presented in Figure S14 is done only on the distances between the first objects around each cluster-center, such that both the *true* coordinates (i.e., the cluster-center-coordinates) and the *measured* coordinates are the same for the ‘test-set’ in the 5 datasets.

SI.G Tree visualization

Through *Bonsai-scout*, we offer an interactive dashboard with which the *Bonsai* tree can be visualized and explored interactively. We will here shortly discuss some core aspects of the available visualizations.

SI.G.1 Tree layout algorithms

In *Bonsai-scout*, we offer three layout-types that can be used to plot the tree in two dimensions: the ladderized dendrogram, the equal-angle layout, and the equal-daylight layout. The full algorithms can be found in the *Bonsai* GitHub-repository [24] in the file `bonsai_scout/my_tree_layout.py`.

The *dendrogram layout* may be the most common way to visualize trees and does not require a description. We here choose to ladderize the dendrogram, which means that, from any node, we choose to sort the branches from top-to-bottom based on how many leaves there are below those branches.

The equal-angle and equal-daylight algorithm were proposed by Felsenstein in his excellent book *Inferring phylogenies* on pages 578 to 584 [27]. These are circular layouts that are especially suitable for unrooted trees like the one that *Bonsai* produces. The equal-angle algorithm works well for trees of all sizes, and will never lead to crossing edges. The equal-daylight algorithm can be

used to improve the equal-angle tree by more homogeneously distributing the leaves over the circle. Although the equal-daylight algorithm leads to considerably better results for small trees, it can become computationally intense for large trees, and can, in the implementation suggested originally, even lead to crossing edges. We have re-implemented the equal-daylight algorithm to streamline its computation and to avoid crossing edges, but on large trees the benefits are modest. Therefore, we only offer the equal-daylight layout for trees with less than 2000 nodes.

SI.G.2 The interpretation of tree distances

It is important to note that distances between tree-nodes should always be measured by adding up the branch lengths along the path that connects the nodes on the tree, rather than by measuring the 2-dimensional distance between the positions of the nodes in the figure. This has a number of consequences. First, as long as we keep the tree topology and the branch lengths fixed, we can visualize the tree in different ways without changing the tree's content. We benefit from this by offering different tree layouts. Second, the tree is independent of the order in which branches are connected to a node. For example, when a node has two downstream nodes, say A, and B, the tree-likelihood is independent of ordering these as AB, or as BA, even though it affects the 2-dimensional visualization. To aid users in gathering intuition for the effects of these meaningless changes, we offer a "Layout tweaking"-tab in *Bonsai-scout*, where one can change the ordering of downstream branches around a selected node.

SI.G.2.1 Distances in the dendrogram layout

One of the layouts that is offered in *Bonsai-scout* is the dendrogram layout. In the dendrogram layout, it should be noted that vertical distances are uninformative for the actual distances between tree-nodes. Indeed, in the dendrogram layout, distances should be measured by *only summing horizontal branch lengths* along the path on the tree between the objects.

SI.G.2.2 Distances on a disk with hyperbolic geometry

For trees that have large variation in the branch lengths, it can be difficult to visualize the whole tree while still being able to resolve details on a part of the tree. To accommodate this, we project the tree on a hyperbolic disk on which distances near the edge of the disk get more and more compressed. To offer the user some intuition for this, when we visualize a hyperbolic disk, we always include a number of equal-sized squares on the background that appear to get smaller towards the edges of the disk. Branch lengths can thus be visually measured by how many of the equal-sized squares the branch crosses.

Technically, the projection to the hyperbolic disk uses a mapping from the 2-dimensional reals to the 2-dimensional unit ball. The projection is dependent on two parameters, an origin $\mathbf{x}_0 = (x_0, y_0)$ and a zoom z . Given the position of any 2-dimensional point (x, y) , we first subtract the origin, and then scale by the zoom: $(x, y) \rightarrow (\bar{x}, \bar{y}) = (z(x - x_0), z(y - y_0))$. Next, we switch to polar coordinates, (r, θ) , such that $\bar{x} = r \cos(\theta)$, and $\bar{y} = r \sin(\theta)$. The hyperbolic mapping leaves θ unchanged, while it maps r to $\frac{r}{1 + \sqrt{1 + r^2}}$. Note that this mapping sends an infinitely large r to 1, and the origin ($r = 0$) to itself.

SI.H Analysis of football player statistics

To illustrate that *Bonsai* can robustly identify the structure of any high-dimensional dataset, looked for a dataset that is maximally different in nature from the single-cell omics datasets we looked at so far. We decided to apply *Bonsai* to a dataset with statistics of professional football players [15]. Although somewhat tongue-in-cheek, this application is illustrative because these non-scientific data are very different in nature, are not expected to fall on a tree, but are nonetheless familiar enough to allow us to judge whether the results make sense.

As shown in Suppl. Fig. S32, even though we do not use any specifically tailored data pre-processing, *emphBonsai*'s tree identified 12 well-separated groups of players and visualization of marker features identified by *Bonsai-scout* (Suppl. Figs. S33-S36) shows that these groups indeed correspond to players that fulfill distinct known roles in their teams. Additionally, *Bonsai* identified some players as clear outliers, indicating that they show exceptional statistics, and, remarkably, we find that these were indeed some of the highest-reputed football players in the season of 2021-2022. In summary, although we are not claiming that *Bonsai*'s analysis identifies new structure in football player statistics that is not already known to experts, the results do support that *Bonsai* automatically identifies subtle and meaningful structures in a dataset of a wildly different nature.

References

1. Zappia, L., Phipson, B. & Oshlack, A. Exploring the single-cell RNA-seq analysis landscape with the scRNA-tools database. *PLOS Comput. Biol.* (2018).
2. Svensson, V., da Veiga Beltrame, E. & Pachter, L. A curated database reveals trends in single-cell transcriptomics. *Database* (2020).
3. Breda, J., Zavolan, M. & van Nimwegen, E. Bayesian inference of gene expression states from single-cell RNA-seq data. *Nat. Biotechnol.* (2021).
4. Stoeckius, M. *et al.* Simultaneous epitope and transcriptome measurement in single cells. *Nat. Methods* (2017).
5. Tarhan, L. *et al.* Single Cell Portal: an interactive home for single-cell genomics data. *bioRxiv* (2023).
6. Li, M. *et al.* DISCO: a database of Deeply Integrated human Single-Cell Omics data. *Nucleic Acids Res* (2022).
7. Papatheodorou, I. *et al.* Expression Atlas update: from tissues to single cells. *Nucleic Acids Res* (2020).
8. Franzén, O., Gan, L.-M. & Björkegren, J. L. M. PanglaoDB: a web server for exploration of mouse and human single-cell RNA sequencing data. *Database (Oxford)* (2019).
9. Schaum, N. *et al.* Single-cell transcriptomics of 20 mouse organs creates a Tabula Muris. *Nature* (2018).
10. Consortium, T. T. S. *et al.* The Tabula Sapiens: A multiple-organ, single-cell transcriptomic atlas of humans. *Science* (2022).

11. Bastian, F. B. *et al.* The Bgee suite: integrated curated expression atlas and comparative transcriptomics in animals. *Nucleic Acids Res* (2021).
12. Packer, J. S. *et al.* A lineage-resolved molecular atlas of *C. elegans* embryogenesis at single-cell resolution. *Science* (2019).
13. Wolock, S. L., Lopez, R. & Klein, A. M. Scrublet: Computational Identification of Cell Doublets in Single-Cell Transcriptomic Data. *Cell Systems* (2019).
14. Stuart, T. *et al.* Comprehensive integration of single-cell data. *Cell* (2019).
15. Kaggle - 2021-2022 Football Player Stats, Version 3.
16. Grün, D., Kester, L. & van Oudenaarden, A. Validation of noise models for single-cell transcriptomics. *Nat. Methods* (2014).
17. Marinov, G. K. *et al.* From single-cell to cell-pool transcriptomes: stochasticity in gene expression and RNA splicing. *Genome Res* (2014).
18. Padovan-Merhar, O. *et al.* Single mammalian cells compensate for differences in cellular volume and DNA copy number through independent global transcriptional mechanisms. *Molecular Cell* (2015).
19. Stegle, O., Teichmann, S. A. & Marioni, J. C. Computational and analytical challenges in single-cell transcriptomics. *Nat Rev Genet* (2015).
20. Tang, W. *et al.* bayNorm: Bayesian gene expression recovery, imputation and normalization for single-cell RNA-sequencing data. *Bioinformatics* (2020).
21. Gillespie, D. T. *Markov processes: an introduction for physical scientists* (Elsevier, 1991).
22. Gardiner, C. *Stochastic Methods: A Handbook for the Natural and Social Sciences* (Springer, 2009).
23. Saitou, N. & Nei, M. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution* (1987).
24. De Groot, D. H., Morillo Leonardo, S. X., Pachkov, M. & van Nimwegen, E. Bonsai-data-representation. Source code. GitHub. <https://zenodo.org/records/20370956>.
25. Baron, M. *et al.* A single-cell transcriptomic map of the human and mouse pancreas reveals inter- and intra-cell population structure. *Cell Syst* (2016).
26. Zeisel, A. *et al.* Cell types in the mouse cortex and hippocampus revealed by single-cell RNA-seq. *Science* (2015).
27. Felsenstein, J. *Inferring Phylogenies* (Sinauer, 2003).