nature protocols

https://doi.org/10.1038/s41596-025-01314-w

Protocol

Check for updates

CellRank: consistent and data view agnostic fate mapping for single-cell genomics

Philipp Weiler [1,2,3,4] & Fabian J. Theis [1,2,3]

Abstract

Single-cell RNA sequencing quantifies biological samples at an unprecedented scale, allowing us to decipher biological differentiation dynamics such as normal development or disease progression. As conventional single-cell RNA sequencing experiments are destructive by nature, reconstructing cellular trajectories computationally is an essential aspect of analysis pipelines. To infer trajectories in a consistent and scalable manner, we have developed CellRank. In its first iteration, CellRank quantitatively recovered trajectories from RNA velocity estimates and transcriptomic similarity. Given these data views, CellRank constructed a cell–cell transition matrix, inducing a Markov chain to automatically infer terminal states and describe their lineage formation. However, CellRank did not enable incorporating complementary data views such as experimental time points, pseudotime or stemness potential. To facilitate these and future views, CellRank 2 generalizes CellRank's trajectory inference framework to multiview single-cell data, leading to a general and scalable framework for cellular fate mapping. Overall, the CellRank framework enables the consistent quantification of cellular fate, combining complementary views and analyzing lineage priming consistently. Here we provide detailed protocols on how to run exemplary CellRank analyses at scale and across different data views. Using CellRank requires basic apprehension and knowledge of single-cell omics data and the Python programming language.

Key points

- This Protocol describes the inference of cell fate from single-cell RNA sequencing data using CellRank. Cell–cell transition probabilities and terminal states are inferred to study fate priming in the corresponding lineages.
- CellRank is consistent and scalable and—unlike other methods, which focus on a single source of information—incorporates complementary views of the data such as pseudotime, RNA velocity and experimental time points while facilitating newly emerging ones.

Key references

Lange, M. et al. *Nat. Methods* **19**, 159–170 (2022): https://doi.org/10.1038/s41592-021-01346-6

Weiler, P. et al. *Nat. Methods* **21**, 1196–1205 (2024): https://doi.org/10.1038/s41592-024-02303-9

Klein, D. et al. *Nature* **638**, 1065–1075 (2025): https://doi.org/10.1038/s41586-024-08453-2

Lange, M. et al. *Genome Biol.* **25**, 1–38 (2024): https://doi.org/10.1186/s13059-024-03422-4

[1]Institute of Computational Biology, Helmholtz Center, Munich, Germany. [2]School of Computation, Information and Technology, Technical University of Munich, Munich, Germany. [3]TUM School of Life Sciences, Technical University of Munich, Munich, Germany. [4]Present address: Program for Computational and Systems Biology, Sloan Kettering Institute, Memorial Sloan Kettering Cancer Center, New York, NY, USA. e-mail: fabian.theis@helmholtz-munich.de

Introduction

Single-cell sequencing has revolutionized how we investigate biological systems by enabling the study of rare cell states[1] or revealing previously obscured ones[2,3], mapping cell states of entire tissues and organs[4–9] and reconstructing complex differentiation trajectories[5,10–12]. Trajectory inference, however, has often been limited by methods designed for specific data types that do not generalize to newly emerging data views (see Box 1 for the most common approaches) or do not describe the long-term behavior of individual cells[13–15].

Integrating novel and orthogonal information into trajectory inference is essential for describing biological processes faithfully. However, the need to adapt or even design specific methods de novo slows down data analysis and makes comparing results across models challenging. To unlock the full potential of current and upcoming experimental innovations, we have developed the CellRank framework[14,16] as a data-view agnostic framework to infer cellular fate from any prior vector field describing cellular change and to combine complementary views. We initially developed CellRank to infer cellular fate from RNA velocity and transcriptomic similarity. However, because RNA velocity is not universally applicable and, conversely, alternative sources of information exist, we extended and generalized our framework in the form of CellRank 2 to facilitate scalable, modular and data view-agnostic fate mapping. Importantly, our framework scales to atlas-sized sample collections and accommodates newly emerging sources of information, making it applicable to upcoming datasets and novel dynamic inference approaches.

Overview of the procedure

The CellRank framework naturally divides itself into three parts: kernels for estimating cell–cell transition probabilities, estimators for analyzing them and analysis tools for downstream tasks like putative driver prediction (Fig. 1). So far, we have introduced five kernels, each leveraging different sources of information: the VelocityKernel infers state change probabilities by comparing RNA velocity and empirical state change estimates; the ConnectivityKernel maps cell–cell similarities to cell–cell transitions; the PseudotimeKernel biases undirected cell–cell similarity graphs toward increasing pseudotime to define putative future cell states probabilistically; the CytoTRACEKernel estimates a stemness score, aligning cells along the differentiation trajectory followed by our pseudotime approach; and the RealTimeKernel uses

BOX 1

Common approaches for inferring cellular trajectories

Pseudotime

Pseudotime algorithms align single-cell data along a one-dimensional manifold to reconstruct dynamical processes from snapshot data, commonly relying on prior information to direct the inference. Immature cells score a low pseudotime that increases with maturity[13,26,39,40,63,64].

RNA velocity

RNA velocity relates unspliced and spliced mRNA through a dynamical model describing splicing dynamics; RNA velocity can be inferred from single-cell transcriptomics data because common experimental assays contain the relevant information for reconstructing unspliced and spliced counts. Various inference algorithms have been proposed, each posing method-specific assumptions[30,35,36,46].

OT

OT leverages time point information to match cells at one time point with their putative future state at a later time point in a probabilistic fashion. On a high level, OT transforms the original distribution into the distribution of progenitor states in an optimal manner[15,24,60,65].

Metabolic labeling

Metabolic labeling experiments label newly transcribed mRNA molecules with nucleotide analogs, thereby introducing temporally related quantities such as unspliced and spliced mRNA. Compared with classical sequencing data from distinct time points, metabolic labeling uses shorter time scales, allowing it to reveal biological mechanisms and distinguish regulatory effects[16,31,51,66–69].

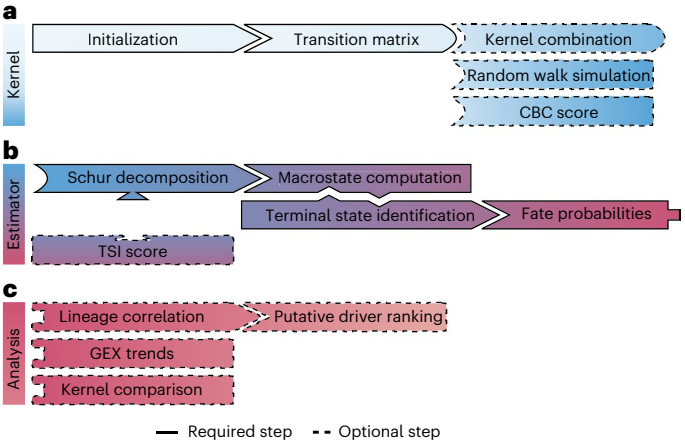


Fig. 1 | CellRank is a modular and scalable framework for cellular fate mapping. a, CellRank divides naturally into kernels, estimators and downstream analysis tools. Kernels are initialized using data and view-specific information to compute a cell–cell transition matrix. Optionally, these matrices may be combined from different views and the induced random walks simulated for a preliminary understanding of the induced dynamics. The CBC score evaluates how faithfully a kernel recapitulates known cell state transition. **b**, Estimators take a cell–cell transition matrix to ultimately compute fate probabilities. Our GPCCA[19] estimator finds these probabilities by coarse-graining the transition matrix to essential states—the so-called macrostates—via a Schur decomposition. Following the macrostate identification, terminal states are either automatically inferred or manually defined; macrostate and terminal state computation may be repeated multiple times until a satisfactory decomposition is identified. Given an estimator, the TSI score systematically quantifies the recovery of known terminal states compared with the number of macrostates used. **c**, Following the fate quantification, the results can be used to identify putative lineage drivers as genes whose GEX correlates with fate probabilities or fit GEX patterns with generalized additive models where a cell's fate weights its contribution to each lineage. Comparing kernel performance is another aspect incorporating both kernel (CBC) and estimator (TSI) level methods.

time-resolved single-cell data by combining optimal transport (OT)-based cell change estimates across time with state shifts inferred from asynchronous cell state changes within time points.

Every CellRank workflow uses kernels to inform an estimator that ultimately enables downstream analyses—the specific flavors of these three general concepts are not predefined, though, making CellRank modular. As such, analyses can also include subsets of the canonical kernel-estimator workflow, that is, studying cell transitions with custom approaches or analyzing precomputed state transitions is possible, for example. Overall, CellRank's modular and scalable design makes it a generic and versatile tool for single-cell sequencing data analysis.

Input requirements

CellRank operates on cell-specific state change estimates, but different kernels require additional data types beyond gene expression (GEX) captured by every single-cell RNA sequencing (scRNA-seq) experiment (Box 2); both the VelocityKernel and PseudotimeKernel rely on a precomputed cell–cell similarity graph, but the VelocityKernel relates it to precomputed, cell-specific velocities and the PseudotimeKernel to a precomputed pseudotime; the RealTimeKernel requires the experimental time point that each observation stems from. The downstream CellRank workflow is then independent of the input data type.

Computing cell–cell transition probabilities with kernels (Stage 1)

As a first step of CellRank-based data analyses, a kernel either computes cell–cell transition probabilities on the basis of problem-specific data or collects a precomputed transition matrix (Fig. 1a and Box 2). The PseudotimeKernel and VelocityKernel are agnostic to the inference method for generating pseudotime and velocity estimates respectively, only requiring the outputs thereof; the PrecomputedKernel collects and prepares a priori estimated transition matrices for inferring the cellular fate with CellRank. To guide kernel choice, previous work[16] provides a decision tree, because kernels can require kernel-specific input—such as experimental time points for the RealTimeKernel, for example—or the underlying data views

BOX 2

Details on kernel inputs

ConnectivityKernel

The ConnectivityKernel requires a precomputed, symmetric cell–cell similarity graph. The kernel is independent of the computation of the similarly matrix and only requires it as a field in the obsp slot of the AnnData object.

PseudotimeKernel

Although each pseudotime method has its own assumptions, CellRank works with any pseudotime. The pseudotime must be saved as a column in the obs slot of the AnnData object and the column name specified when initializing the PseudotimeKernel. In addition, the PseudotimeKernel requires a precomputed, symmetric cell–cell similarity graph, saved in the obsp slot of the AnnData object.

CytoTRACEKernel

Inferring the CytoTRACE score as a stemness potential does not require any additional input beyond GEX counts. The kernel converts these cell scores into a cell–cell transition matrix, by biasing an undirected cell–cell similarity graph, saved in the obsp slot of the AnnData object, into the direction of decreased potential.

VelocityKernel

CellRank decouples the inference of cell–cell transition probabilities on the basis of cellular velocity estimates from the actual velocity inference. Estimating cell–cell transition probabilities requires both cell-specific GEX representation and velocity estimates, saved either in the layers or obsm slot of the AnnData object, and a precomputed, symmetric cell–cell similarity graph, saved in the obsp slot; the corresponding field names are passed to the VelocityKernel upon initialization.

RealTimeKernel

The RealTimeKernel connects cells across distinct experimental time points using OT. It reads the experimental time point of each cell from a corresponding column in the obs slot of the AnnData object; the corresponding column name is passed to the kernel when initializing it.

PrecomputedKernel

Users computing cell–cell transition matrices independent of CellRank can still rely on the CellRank framework for trajectory inference through the PrecomputedKernel. The kernel requires the transition matrix in the obsp slot of the AnnData object.

pose problem-specific constraints—such as the time scale cellular dynamics on the order of splicing dynamics for RNA velocity, for instance.

(Optional) Combining kernels

Different data views may provide alternative insight into the underlying biological process. To benefit from these complementary system descriptions, multiple kernels can be combined through a weighted average of their transition matrices. Although CellRank requires at least one cell–cell transition matrix to function, combining multiple is optional. Previous analyses benefited from this step in terms of correctly identifying all terminal states and numerical stability, as one view may regularize another[14,16–18].

Inferring initial and terminal states (Stage 2)

CellRank models average cellular behavior, assuming incremental, memoryless change along the phenotypic manifold. Therefore, CellRank collects the cell–cell transition probabilities in a cell–cell transition matrix that defines a Markov chain (Box 3). Given any cell–cell transition matrix, an estimator analyzes the induced Markov chain to infer the initial and terminal states of a biological process, quantify the corresponding fate probabilities and enable further downstream analyses (Fig. 1b). In brief, our generalized Perron cluster cluster analysis (GPCCA)[19] estimator coarse grains the transition matrix to define macro states and transition probabilities between them, thereby defining terminal states and fate probabilities.

Downstream analyses (Stage 3)

Given the terminal states and the fate probabilities toward them, CellRank provides functions for analyzing the overall inference but also underlying fate priming, that is, the gradual commitment of a cell to a specific lineage (Fig. 1c). To help understand the gene dynamics driving state changes and fate priming, CellRank can provide a list of putative lineage drivers by correlating GEX with fate probabilities; ranking these lineage-correlated genes on the basis of

BOX 3

Markov chain-related concepts and definitions

Markov chain
Markov chains describe dynamic processes by connecting a set of states probabilistically. In the context of CellRank, the states are the sequenced cells.

Transition probability
Transition probabilities quantify how likely one cell is the ancestor state of another.

Random walks
Starting from a given cell and moving to one of its putative future states on the basis of the inferred cell–cell transition probabilities defines a random walk. Random walks provide a simulation framework to assess the induced Markov chain.

Macrostate
States and the underlying dynamics described by a Markov chain may be noisy. Macrostates represent relevant biological states, that is, a denoised and coarse-grained version of the original Markov chain.

Initial state
The initial state of a system is the state where the underlying dynamical process starts.

Terminal state
Terminal states are states present in the data for which no progenitor states exist. These states may be fully differentiated cell types or intermediate states for which no future states have been captured.

Fate probability
Fate probabilities describe the probability of a cell to differentiate into a terminal state.

their peak expression can indicate putative regulator–target relationships; and plotting their expression profile along each lineage can help elucidate activation and inhibition patterns. Importantly, the estimates may function as input beyond our framework in a more principled manner as they can inform lineage-associated differential GEX analysis with tradeSeq[20] or the probability mass flow in time[21], for example. Relatedly, to study lineage-specific GEX changes, CellRank fits Gaussian additive models, incorporating both GEX and lineage weights corresponding to fate probabilities. Finally, to compare the performance of different kernels, the terminal state identification (TSI) score, for example, quantifies how faithfully a kernel-derived transition matrix recovers known terminal states and, thus, aids in model selection[16]. If state transitions are known, the kernel-based cross-boundary correctness (CBC) quantifies how accurately the kernel aligns with known state transitions[16]. Although kernels and our estimator compute the CBC and TSI scores, respectively (Fig. 1a,b), we commonly rely on them downstream of each building block of CellRank to assess overall performance (Fig. 1c).

Comparison and compatibility of CellRank 1 and 2

Results found with CellRank 1 are generally compatible with CellRank 2 workflows. Importantly, the methodological concepts for inferring cellular fate from RNA velocity estimates with CellRank 1 are unchanged in CellRank 2. As such, the two versions will yield the same results as long as the analysis leaves all other Python packages unchanged. However, application programming interface (API)-related changes may require updating several variable or field names in the corresponding AnnData object. We provide an overview of the most important changes in the documentation of our software, guiding users to implement these changes to switch to the latest CellRank version.

Applications of the method

We and others have applied CellRank to a diverse set of biological systems and even relied on our framework to incorporate new data modalities: as an example of RNA velocity, CellRank has provided insight into murine pancreatic endocrinogenesis, lung regeneration and mouse embryonic fibroblast reprogramming[14]; using metabolic labels as an alternative source for inferring cell velocities, we have studied murine gut development[16]. We have similarly relied on pseudotime and CytoTRACE estimates to describe human hematopoiesis and embryoid body development, respectively, and the RealTimeKernel has allowed us to identify a putative progenitor population of medullary thymic epithelial cells using information from experimental time points[16]. Similarly, we have connected disease stages of Alzheimer's disease to infer a disease-stage-informed pseudotime and describe changes in the composition of microglia substates over disease progression[22]; other studies have used our CytoTRACEKernel to decipher unique signaling principles of leptomeningeal antitumor immunity[23] or combined pseudotime and RNA velocity estimates to study human heart development and disease in organoid models[18].

As a flexible framework that renders itself applicable to emerging data views, CellRank allowed moscot[24] to seamlessly map cellular state changes across time and space when studying mouse development and moslin[25] to analyze time-resolved single-cell data on the basis of time point and lineage tracing information. Similarly, EpiTrace incorporated open clock-like loci from single-cell assay for transposase-accessible chromatin (scATAC-seq) using sequencing measurements in combination with RNA velocity and stemness estimates to study glutaminergic neuron development in the human fetal brain cortex[17].

Alternative methods

Trajectory inference is a long-standing field of single-cell genomics, but existing methods focus on a single data view: Palantir[13], for example, infers cell–cell transition probabilities on the basis of its inferred pseudotime and the underlying cell–cell similarity graph. Whereas CellRank mimics this approach, the two methods differ in how they infer cellular fate from the induced Markov chain—CellRank follows the GPCCA approach, whereas Palantir assumes that terminal states reside near or on the boundary of the phenotypic manifold and defines them as the intersection of extrema in the stationary distribution of the Markov chain and diffusion components. Alternatively, Slingshot[26] assigns each cell a pseudotime and lineage weight simultaneously by inferring a minimal spanning tree; other methods rely on Gaussian processes[27], self-organizing maps[28] or generalized linear models[29]. Importantly, each method comes with its own sets of assumptions, and not every method can use the full analysis pipeline of state change quantification, fate inference and following downstream analyses. CellRank unifies pseudotime-based fate mapping by decoupling fate inference from pseudotime inference.

Cell velocities provide directed, dynamic information, and different methods have been developed to infer the change. Velocyto[30], for example, was the first method for inferring RNA velocity and quantifies long term behavior by simulating Markov chains and defining a cell's future state on the basis of its most probable state after a fixed number of steps along the chain; the GEX profile, RNA velocity and neighborhood of each cell define the corresponding transition probabilities. Dynamo[31] uses a similar mechanistic model to incorporate metabolic labeling information into velocity inference. To quantify cellular change, Dynamo approximates the continuous vector field first to then identify initial and terminal states as the field's attractors; genes whose expression changes most from the optimal transition path toward a state—the least action path—are postulated putative lineage drivers. Similar to pseudotime inference, each method comes with its own set of assumptions and data requirements and does not generalize to alternative data views.

OT offers a framework for incorporating real time information into trajectory inference by matching cells in one time point with putative progenitor states in the consecutive time point[15,24], defining so-called transport maps. Waddington-OT[15], for example, propagates state changes by multiplying the matrices associated with each transport map. However, OT neglects transitory information within a single time point that arises from the asynchronous unfolding of

biological processes. CellRank combines inter-time point information through any OT method with intra-time point information, thereby combining the two aspects.

Limitations

Data quality and model assumptions inherently limit CellRank's ability to accurately estimate fate probabilities. As such, each kernel has its own limitations:

- RNA velocity inference faces shortcomings stemming from both experimental and computational limitations[16,32,33]: on the experimental side, gene structure biases affect the faithful recovery of unspliced mRNA because polyadenylation and splicing happen mostly simultaneously, and most unspliced counts, thus, stem from internal priming events. On the computational side, accurate model descriptions are unfeasible, and current inference schemes use simplistic models that recapitulate complex systems inadequately[32,34]. Specifically, approaches traditionally assume constant kinetic rates, neglect gene–gene interactions and do not consider chromatin and proteins[30,35,36].
- Pseudotime methods traditionally assume a unidirectional process from less to more differentiated cell states and oftentimes require a user-defined root cell, marking the start of the process. As such, pseudotime inference is challenging on complex or understudied systems that either do not follow an increasing differentiation order[37] or the initial state is unknown[23].
- For time series data, sampling the developmental system under study often and frequently enough to capture the change in GEX is essential. However, choosing the correct spacing between time points is unclear, affecting CellRank's ability to recapitulate the process accurately. If the gap between consecutive time points is too large, CellRank may miss essential changes in transcriptional profiles merely because they have not been captured and observed and, thus, fail to correctly model state transitions.
- CellRank focuses on average cellular behavior to assume memoryless transitions and apply the rich theory of Markov chains. However, this assumption may miss rare cell states and long-term and delayed effects caused by accumulated proteins, for example.

Single-cell sequencing data are high dimensional, sparse and noisy. These characteristic properties make any inference task challenging and directly influence CellRank's performance—the data noise, for example, affects how well cell–cell transition probabilities recapitulate ground truth, and, as a result, automatically identifying the correct number of terminal states can be challenging. However, CellRank does not address these properties itself as it infers cell–cell transition probabilities on the basis of precomputed information. Instead, methods operating upstream of CellRank, that is, tools providing the input for CellRank, handle data sparsity in a method-dependent manner: common RNA velocity approaches smooth scRNA-seq data by averaging expression across neighborhoods[30,35,36,38], pseudotime inference methods commonly rely on low-dimensional or graph representations[13,39,40] and OT-based inference operates on low-dimensional representations such as from principal component analysis (PCA) or latent spaces from deep learning-based methods such as scVI, for example[15,24]. Although CellRank is robust overall, its different analysis steps may require a less stringent study compared with an ideal setting.

Even if data quality were not an issue, CellRank's framework leaves room for improvement: in its current state, CellRank assigns fate probabilities toward terminal states but does not decipher the most probable path of a cell toward them—studying non-tree-like differentiation phenomena such as transcriptional convergence, thus, becomes challenging. CellRank also identifies putative drivers on the basis of correlation but misses causal links. Although we and others have successfully uncovered lineage drivers using this approach, causal descriptions would facilitate in silico predictions and aid in model understanding without the need for experimental follow-ups to validate putative regulators.

The absence of ground truth resulting from the destructive nature of sequencing protocols is a general challenge for the single-cell field. Still, some prior knowledge of the expected cell state change exists for many systems. CellRank-based analyses can benefit from it by using metrics we devised to compare kernel performance both in absolute and relative terms[16]. Although these same metrics may aid in assessing how to weigh kernel combinations, the weighting is global and cell-specific weighting could facilitate the strengths of each data view.

Experimental design

CellRank infers cell–cell transition probabilities and terminal states to assign observations to the corresponding lineages. The canonical workflow consists of defining cell–cell transition probabilities, inferring macrostates of the induced Markov chain and ultimately defining terminal states. Fate probabilities toward these states define the corresponding lineages and help elucidate the inference: the TSI score describes how accurately known terminal states are recovered and the CBC score to what extent known state transitions are recapitulated. CellRank further suggests putative lineage drivers by ranking genes on the basis of the correlation between their expression and fate probabilities; the fate probabilities can further inform Gaussian additive models fit to model GEX change over pseudotime, describing gene behavior within the different lineages.

Procedure 1 guides through a classical CellRank-based analysis flow with the CytoTRACEKernel to describe hematopoiesis by using stemness estimates (Fig. 2a). CellRank allows the inference of a CytoTRACE score to describe stemness at scale, but assessing the accuracy of such estimates is crucial. Procedure 1 exemplifies this step after inferring macrostates, defining terminal states on the basis of them and assigning cell-specific fate probabilities toward them (Fig. 2b,c and Extended Data Fig. 1a). The procedure relies on prior knowledge to evaluate the CytoTRACE-based results in terms of visually assessing the fate probabilities as skewed toward erythroid cells (Extended Data Fig. 1a), highlighting the challenges of recovering all terminal states with the TSI score (Fig. 2d) and computing the CBC score.

Procedure 2 follows a similar workflow to Procedure 1, using the same hematopoiesis data, but uses the PseudotimeKernel to show how CellRank can recover lineage-correlated genes to explain lineage formation in addition to CellRank's inference results. Specifically, Procedure 2 suggests a substate of the existing cell type clusters is biased toward the nonobserved megakaryocyte state (Fig. 3d–f). Procedure 2 also exemplifies how the results of two different kernels can be compared on the basis of the CBC score, highlighting the PseudotimeKernel outperforming the CytoTRACEKernel on this specific dataset.

CellRank is modular, allowing the easy usage of complementary views. Procedures 3 and 4 showcase this feature by inferring cell–cell transition probabilities on the basis of velocity estimates (VelocityKernel) and real-time information (RealTimeKernel), respectively. Procedure 3 uses a dataset of spermatogenesis and also shows how to combine kernels to potentially benefit from complementary views; Procedure 4 uses the RealTimeKernel in the context of hematopoiesis with a time-resolved mouse bone marrow dataset. The consecutive steps shown exclusively in Procedures 1 and 2 could be performed equivalently as they only rely on a kernel, not a specific data view.

Materials

Equipment

Hardware

A laptop, desktop workstation or computer server with an internet connection. The presented protocols were run on a MacBook Pro (macOS Sequoia, Version 15.3) with a 14-core central processing unit and 48 GB of random-access memory.

Software

- Operating system: Linux, MacOS or Windows
- Python version: at least 3.10
- CellRank: the actively maintained open-source code is publicly available via GitHub at https://github.com/theislab/cellrank
- Other Python packages: CellRank automatically installs the latest compatible versions of the third-party packages it depends on and is not restricted to specific package versions. If users encounter compatibility issues, we encourage checking the existing issues on our GitHub page and opening a new one if needed (https://github.com/theislab/cellrank/issues)

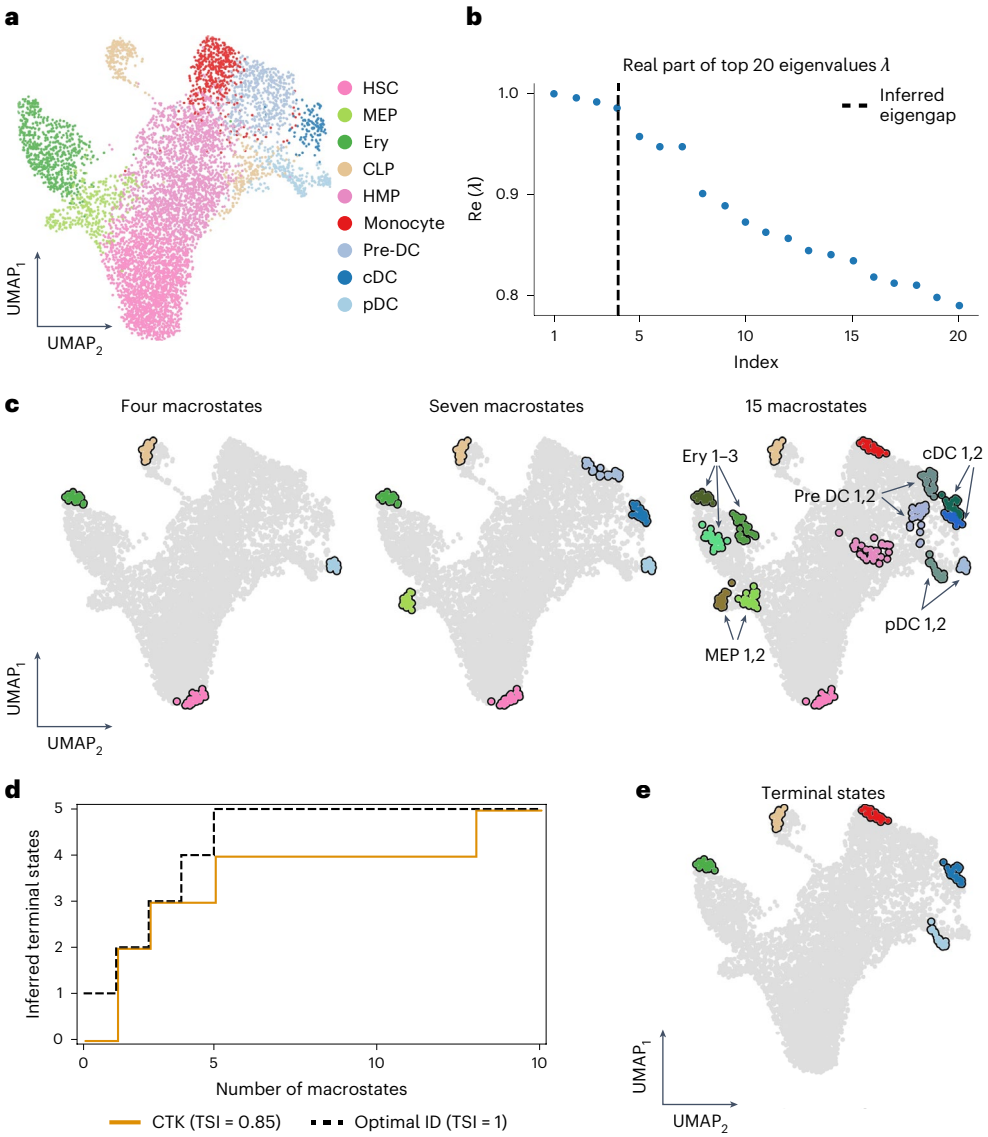


Fig. 2 | Applying CellRank's CytoTRACEKernel to predict fates in human bone marrow development. a, A UMAP embedding of 6,881 CD34$^+$ human bone marrow cells. **b**, The real part Re of the top 20 eigenvalues of the corresponding Schur decomposition, ordered by magnitude. The vertical dashed line indicates the eigengap inferred by CellRank automatically. **c**, A UMAP embedding colored by 4 (left), 7 (middle) and 15 (right) inferred macrostates. The color coding is identical to **a** unless otherwise specified. **d**, A TSI curve for CytoTRACEKernel (solid orange) and an optimal identification (dashed black); an optimal identification finds one additional terminal state for each added macrostate. **e**, The UMAP embedding with the manually set terminal states highlighted; cells colored gray are other cells identified as nonterminal. HSC, hematopoietic stem cells; CLP, common lymphoid progenitors; HMP, hematopoietic multipotent progenitor cells; pre-DC, pre-dendritic cells; pDC, plasmacytoid dendritic cells; cDC, classical dendritic cells; Ery, erythroid; CTK, CytoTRACEKernel; ID, identification.

Data required for general analyses

▲ **CRITICAL** The CellRank-based workflows require kernel-specific and precomputed data as listed below.

- VelocityKernel: GEX matrix, cell-specific velocities and a cell–cell similarity graph
- ConnectivityKernel: GEX matrix and a cell–cell similarity graph
- PseudotimeKernel: GEX matrix, cell-specific pseudotime and a cell–cell similarity graph
- CytoTRACEKernel: GEX matrix and a cell–cell similarity graph
- RealTimeKernel: GEX matrix and experimental time point information

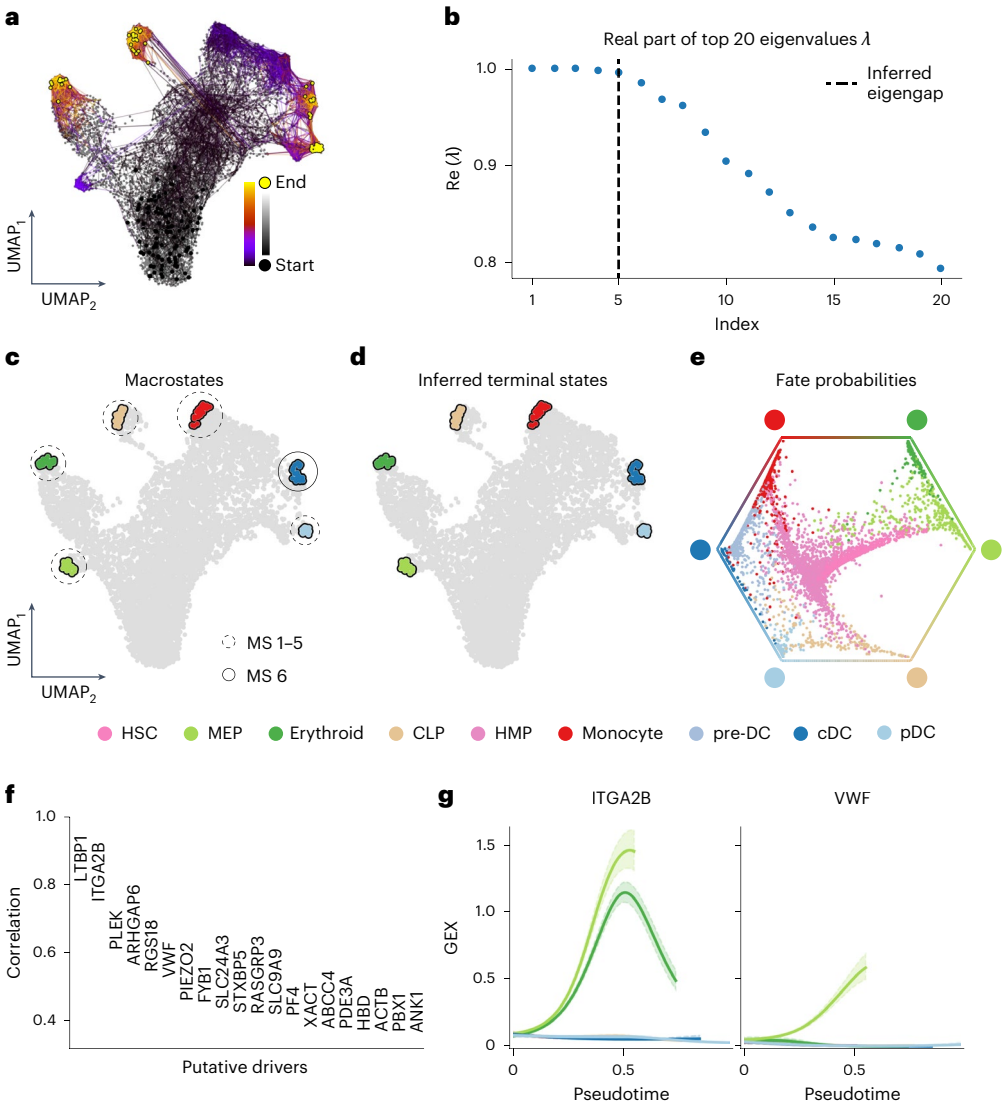


Fig. 3 | The PseudotimeKernel recapitulates human bone marrow development. a, Random walks visualized on the UMAP embedding of the human bone marrow data. Black dots indicate the start of a random walk and consecutive stages of the random walk are connected by an edge; the terminated state of the random walk is highlighted in yellow. **b**, The real part of the top 20 eigenvalues of the corresponding Schur decomposition, ordered by magnitude. The vertical dashed line indicates the eigengap inferred by CellRank automatically. **c**,**d**, UMAP embedding, colored by six (**c**) inferred macrostates and the automatically inferred terminal states (**d**), respectively. The color coding is identical to Fig. 2a, and the cells not associated with a macrostate are colored gray. **e**, The fate probabilities toward each inferred terminal state, visualized in a circular projection: terminal states are arranged uniformly around a unit circle, and each cell's position in the unit circle reflects its fate bias toward a terminal state. Each dot within the polygon is a cell colored by its cell type according to Fig. 2a; dots outside the polygon indicate the corresponding terminal state. **f**, The putative drivers of the MEP lineage, ranked by the correlation between their GEX and fate probability toward the MEP state. **g**, The lineage-specific GEX trends with corresponding 95% confidence intervals of ITGA2B and VWF fitted with generalized additive models to GEX (*y* axis) over pseudotime (*x* axis); the fate probabilities define the weighted contribution of each cell to each lineage. The colors correspond to lineages as in **a**. Re, real part; MS, macrostate; HSC, hematopoietic stem cells; CLP, common lymphoid progenitors; HMP, hematopoietic multipotent progenitor cells; pre-DC, pre-dendritic cells; pDC, plasmacytoid dendritic cells; cDC, common dendritic cells.

Example datasets

▲ **CRITICAL** The datasets used in procedures are available via Figshare at https://doi.org/10.6084/m9.figshare.c.7752290.v1 (ref. 41) and the original studies.

- CD34$^+$ human bone marrow[42]: the preprocessed data include the uniform manifold approximation and projection (UMAP)[43] embedding, cell type labels defined by the original study and pseudotime estimates inferred with Palantir[13].
- Time-resolved murine bone marrow[44]: to speed up the corresponding procedure, we have subsampled the original data to 25% of the cells. The data include the experimental time point each observation has been taken at and the precomputed PCA embedding.
- Spermatogenesis[45]: the data includes the nearest-neighbor graph, smoothed unspliced and spliced counts and the cell-specific velocities. Here, we inferred RNA velocity using scVelo[35], but the presented workflow works with velocity estimates from any inference paradigm, including other RNA velocity inference methods[30,36,46–50] or metabolic labeling-based approaches[16,31,51]; see https://theislab.github.io/cellrank_protocol/metabolic_labeling/inference.html for an example on how to infer RNA velocity on the basis of metabolic labeling data.

Equipment setup

Installation

We support installing CellRank with the conda command line tool (https://docs.conda.io/en/latest/) via the conda-forge channel or from PyPI via pip. For both options, we recommend setting up a Python environment with conda, for example

```
conda create -n crp-py311 python=3.11 --yes && conda activate crp-py311
```

For installing CellRank with Conda use

```
conda install -c conda-forge cellrank
```

For installing CellRank from PyPI use

```
pip install cellrank
```

We provide all presented protocols as Jupyter Notebooks and Python scripts, accompanied by helper functions collected in a dedicated package. To only install it, run

```
pip install git+ https://github.com/theislab/cellrank_protocol
```

or to additionally download all files

```
git clone https://github.com/theislab/cellrank_protocol
cd cellrank_protocol
# To install packages required for running the Jupyter notebooks run
# pip install -e ".[jupyter]"
pip install -e .
```

To run the following procedures in a Jupyter notebook, install the relevant Python packages either through the cloned reproducibility repository or manually

```
pip install jupyterlab ipywidgets
python -m ipykernel install --user --name crp-py311 --display-name
"crp-py311"
```

Data

Download the data

```
mkdir -p data
cd data
```

```
mkdir -p bone_marrow/processed
wget https://figshare.com/ndownloader/files/53394941 -O bone_marrow/
processed/adata.h5ad

mkdir -p spermatogenesis/processed
wget https://figshare.com/ndownloader/files/53395040 -O spermatogenesis/
processed/adata.h5ad

mkdir -p larry/processed
wget https://figshare.com/ndownloader/files/55781036 -O larry/processed/
adata.zarr.zip
```

Library imports

To run the following procedures, import all necessary packages and define their abbreviations; to visualize results easily, Jupyter notebooks can be used, but standard Python scripts can be run instead as well.

```
from tqdm import tqdm

import numpy as np
import pandas as pd
from scipy.stats import ttest_ind

import matplotlib.pyplot as plt
import mplscience
import seaborn as sns
from matplotlib.patches import Patch

import anndata as ad
import cellrank as cr
import scvelo as scv

from crp import DATA_DIR, FIG_DIR
```

Data processing

Here, we only discuss analysis steps involving CellRank instead of including data preprocessing and RNA velocity inference, for example. However, for each procedure, we provide the corresponding steps that generate the presented data as Jupyter Notebooks and Python scripts. We also refer to the CellRank tutorials (https://cellrank.readthedocs.io/en/stable/notebooks/tutorials/index.html) and dedicated reviews[38,52] for more examples of how to infer RNA velocity, pseudotime and CytoTRACE score or use OT.

Procedure 1: developmental potential-based fate mapping

Read and prepare data

1. Load GEX data into the AnnData[53] format: as a first example, we demonstrate our protocol by using the CytoTRACEKernel to study hematopoiesis using CD34$^+$ human bone marrow cells[42] (Fig. 1a).

```
adata = ad.io.read_h5ad(DATA_DIR / "bone_marrow" / "processed" /
"adata.h5ad")
```

2. Define constants used during this procedure: define the expected terminal states and known state transitions. The labels correspond to mature cell types and ground truth state transitions present in the data and assigned by the original study, that is, before running CellRank. Update these labels when working with a different dataset accordingly.

```
TERMINAL_STATES = ["Ery", "CLP", "cDC", "pDC", "Mono"]
STATE_TRANSITIONS = [
    ("HSC", "MEP"),
    ("HSC", "HMP"),
    ("HMP", "Mono"),
    ("HMP", "CLP"),
    ("HMP", "DCPre"),
    ("DCPre", "pDC"),
    ("DCPre", "cDC"),
]
```

3. Compute imputed GEX counts to compute CytoTRACE score: for each cell, smooth its GEX by taking the average GEX of similar cells defined by its neighborhood in the precomputed *k* nearest-neighbor graph. The Python package scvelo computes this value via the `moments` function using normalized spliced and unspliced counts. Here, we do not work with old and nascent mRNA, so we define the corresponding layers as the normalized total counts stored in the `X` attribute, instead.

```
adata.layers["spliced"] = adata.X.copy()
adata.layers["unspliced"] = adata.X.copy()

scv.pp.moments(adata, n_pcs=None, n_neighbors=None)
```

Stage 1: estimate cell–cell transition probabilities

4. Initialize kernel: define kernel and compute CytoTRACE score. The CytoTRACE score expects the AnnData object containing the GEX data to compute the CytoTRACE score.

```
ctk = cr.kernels.CytoTRACEKernel(adata)
ctk.compute_cytotrace()
```

5. Compute transition matrix: each kernel computes the transition matrix with the `compute_transition_matrix` function.

```
ctk.compute_transition_matrix(threshold_scheme="soft", nu=0.5)
```

Stage 2: infer terminal states

6. Initialize estimator: estimators take kernels as input to infer terminal states. Here, the GPCCA estimator is initialized with our CytoTRACEKernel.

```
estimator = cr.estimators.GPCCA(ctk)
```

7. Compute Schur decomposition: GPCCA infers macrostates on the basis of the Schur decomposition of eigenpairs[19] (Fig. 2b).

```
estimator.compute_schur(n_components=20)
```

8. (Optional) Plot top eigenvalues of Schur decomposition: an increased eigengap—the difference between ordered eigenvalues—between the corresponding eigenvectors suggests a potential number of macrostates. Note that CellRank proposes a number of

macrostates on the basis of a heuristic that tries to identify a plausible large eigengap automatically (Fig. 2b). In a perfect setup, the eigengap would coincide with the actual number of terminal states, but as single-cell sequencing data are noisy both in terms of gene signatures of each cell and capturing the entire phenotypic manifold itself, the eigengap merely aids in guiding the user in choosing the number of macrostates; users may have to follow an iterative approach exemplified in the following two steps.

```
estimator.plot_spectrum(real_only=True)
```

9. Compute macrostates: this step requires specifying how many macrostates to compute. In our example dataset, an eigengap is observed after the fourth eigenvalue that CellRank's heuristic also identified (Fig. 2b). Hence, four macrostates are computed and visualized in the UMAP embedding (Fig. 2c). Pass the column name that includes the cell type labels to associate the macrostates to; plotting the states is insightful but optional. Note that this computation is only a proof of concept here because we already know that the dataset includes five terminal states, and we, thus, require at least five macrostates. As the identified macrostates do not cover the expected terminal states, we recompute seven as we observe the next increased eigengap after the corresponding eigenvectors, but we still fail to recover all terminal states (Fig. 2c). To include a monocyte-labelled macrostate, we follow the same reasoning and finally increase the number of macrostates to 15 (Fig. 2c).

```
estimator.compute_macrostates(4, cluster_key="celltype")
estimator.plot_macrostates(
  which="all", basis="umap", legend_loc="right", title="", size=100
)

estimator.compute_macrostates(7, cluster_key="celltype")
estimator.plot_macrostates(
  which="all", basis="umap", legend_loc="right", title="", size=100
)

estimator.compute_macrostates(15, cluster_key="celltype")
estimator.plot_macrostates(
  which="all", basis="umap", legend_loc="right", title="", size=100
)
```

◆ **TROUBLESHOOTING**

10. (Optional) Compute TSI score: in our example data, to ensure that 15 macrostates are sufficient to include each terminal state we are looking for, compute the TSI curve. This computation is an alternative to increasing the number of macrostates until all terminal states are included. An estimator can conveniently plot the TSI curve (Fig. 2d), which is saved to compare it with another approach later on.

```
cluster_key = "celltype"

tsi_score = estimator.tsi(
  n_macrostates=15,
  terminal_states=TERMINAL_STATES,
  cluster_key=cluster_key
)
print(f"TSI score: {tsi_score:.2f}")

estimator.plot_tsi()
```

```
estimator._tsi.to_df().to_parquet(
  DATA_DIR / "bone_marrow" / "results" / "ct_tsi.parquet"
)
```

11. Predict or set terminal states: terminal states can either be set manually or inferred automatically on the basis of a predefined heuristic; the stability of a macrostate, that is, the probability of a Markov chain not leaving it, and eigengaps are possible criteria. In this example, terminal states are set manually as the first macrostates associated with the known terminal states; for readability, the macrostates are also renamed by dropping the identifying suffix.

```
# To print the names of inferred macrostates, run
# print(
#   "Identified macrostates: "
#   f"{', '.join(estimator.macrostates.cat.categories.sort_values())}"
#)

estimator.set_terminal_states(["Ery_1", "CLP", "pDC_1", "cDC_1", "Mono"])
estimator.rename_terminal_states(
  {"Ery_1": "Ery", "pDC_1": "pDC", "cDC_1": "cDC"}
)

# Set color of terminal states to the color of the corresponding cell type
celltype_palette = dict(
  zip(
    adata.obs["celltype"].cat.categories, adata.uns["celltype_colors"]
  )
)
terminal_state_colors = [
    celltype_palette[terminal_state]
    for terminal_state in estimator.adata.obs["term_states_fwd"].cat.
categories
]
estimator._term_states = estimator._term_states.set(
    colors=terminal_state_colors
)
```

12. (Optional) Plot terminal states: similar to visualizing the macrostate, the terminal states can be plotted (Fig. 2e).

```
# The arguments 'legend_loc', 'title' and 'size' only have aesthetic
relevance
estimator.plot_macrostates(
    which="terminal", basis="umap", legend_loc="right", title="",
size=100
)
```

13. Compute fate probabilities: to quantify cellular trajectories, assign each observation a fate probability toward the terminal states.

```
estimator.compute_fate_probabilities()
```

◆ **TROUBLESHOOTING**

14. (Optional) Plot fate probabilities: to gain an intuitive understanding of the assigned fate probabilities, for each terminal state, color each observation by the probability of that

cell differentiating into this terminal state. In our example data, we observe that the fate movement toward several cell states violates the expected behavior, such as high fate bias toward the erythroid lineage for almost all cells (Extended Data Fig. 1a).

```
# The argument 'same_plot=False' makes the function generate
# one plot per terminal state
estimator.plot_fate_probabilities(
    same_plot=False, basis="umap", ncols=5
)
```

Stage 3: downstream analyses

15. (Optional) Compute CBC: to quantitatively describe how well the inferred trajectories recapitulate ground truth state transitions, compute the CBC. This step requires pairs of known progenitor–ancestor states, the observation column assigning cells to the respective states and a low-dimensional representation in which to compute the CBC. To compare the performance of the CytoTRACEKernel to an alternative approach, save the CBC result.

```
cluster_key = "celltype"
rep = "X_pca"

# Collect CBCs for each cell state transition in a Pandas DataFrame,
where
# each row corresponds to the CBC of a cell in the source population
with cells
# from the target state in its neighborhood
ctk_cbc = []
for source, target in tqdm(STATE_TRANSITIONS):
  _cbc = ctk.cbc(source=source, target=target, cluster_key=cluster_
key, rep=rep)

  ctk_cbc.append(
    pd.DataFrame(
        {
        "state_transition": [f"{source} - {target}"] * len(_cbc),
        "cbc": _cbc,
      }
    )
  )
ctk_cbc = pd.concat(ctk_cbc)

# This step is only needed to compare with another approach
cbc.to_parquet(
    DATA_DIR / "bone_marrow" / "results" / "ct_cbc.parquet"
)
```

Procedure 2: pseudotime-based fate mapping

Read data and prepare data

1. Load processed GEX data into the AnnData[53] format: we again demonstrate our protocol to study the hematopoietic system using CD34$^+$ human bone marrow cells[42] but relying on a pseudotime with the PseudotimeKernel.

```
adata = ad.io.read_h5ad(
    DATA_DIR / "bone_marrow" / "processed" / "adata.h5ad"
)
```

2. Define constants used during this procedure: define the expected terminal states and known state transitions; the labels correspond to mature cell types and ground truth state transitions present in the data and assigned by the original study, that is, before running CellRank. Update these labels when working with a different dataset accordingly.

```
TERMINAL_STATES = ["Ery", "CLP", "cDC", "pDC", "Mono"]

STATE_TRANSITIONS = [
    ("HSC", "MEP"),
    ("MEP", "Ery"),
    ("HSC", "HMP"),
    ("HMP", "Mono"),
    ("HMP", "CLP"),
    ("HMP", "DCPre"),
    ("DCPre", "pDC"),
    ("DCPre", "cDC"),
]
```

Estimate cell–cell transition probabilities

3. Initialize kernel: the PseudotimeKernel expects the AnnData object to contain the GEX data and the identifier under which the pseudotime is saved.

```
ptk = cr.kernels.PseudotimeKernel(
  adata,
  time_key="palantir_pseudotime"
)
```

4. Compute transition matrix.

```
ptk.compute_transition_matrix(threshold_scheme="soft")
```

5. (Optional) Plot random walks: for a preliminary understanding of the dynamics, random walks can be generated from the induced Markov chain and visualized (Fig. 3a). If the initial cell state is known, observations from it can be used as the start of the random walks; for hematopoiesis, stem cells form the most immature state.

```
ptk.plot_random_walks(
    start_ixs={"celltype": "HSC"},
    basis="umap",
    seed=0,
    dpi=150,
    size=30
)
```

Alternatively, random walks can be started from cells with a low pseudotime.

```
ptk.plot_random_walks(
    start_ixs={"palantir_pseudotime": [0, 0.1]},
    basis="umap",
    seed=0,
```

```
    dpi=150,
    size=30
)
```

Stage 2: infer terminal states

6. Initialize estimator: initialize the GPCCA estimator by passing the kernel to it.

```
estimator = cr.estimators.GPCCA(ptk)
```

7. Compute Schur decomposition.

```
estimator.compute_schur()
```

8. (Optional) Plot top eigenvalues of Schur decomposition: an increased eigengap between the corresponding eigenvectors suggests a potential number of macrostates (Fig. 3b).

```
estimator.plot_spectrum(real_only=True)
```

9. Compute macrostates: this step requires users to specify how many macrostates to compute. In our example dataset, we compute five macrostates, as there is an increased eigengap after the fifth eigenvalue (Fig. 3b). We also pass the column name that includes the cell type labels to associate the macrostates to and plot the states to see which cell states the estimator defines as macrostates; this last step is optional.

```
estimator.compute_macrostates(5, cluster_key="celltype")
estimator.plot_macrostates(
    which="all",
    basis="umap",
    legend_loc="right",
    title="",
    size=100
)
```

As the five computed macrostates do not include the cDC state but the megakaryocyte/erythroid progenitor (MEP) state instead, we compute six macrostates, recovering all expected terminal states (Fig. 3c).

```
estimator.compute_macrostates(6, cluster_key="celltype")
estimator.plot_macrostates(
    which="all",
    basis="umap",
    legend_loc="right",
    title="",
    size=100
)
```

◆ **TROUBLESHOOTING**

10. Predict or set terminal states: this step relies on the automatic identification in a first iteration (in contrast to Procedure 1, where they were directly set manually). By default, the estimator defines any macrostate with a stability exceeding 0.96 as terminal.

```
estimator.predict_terminal_states()
```

11. (Optional) Plot terminal states: in our example dataset, this step shows that CellRank has identified six states as terminal, including the MEPs which we assumed as transitional. Although

we expected five terminal states on the basis of the provided cell type annotation (Fig. 3d), the following downstream analyses will show that including the MEP state is biologically meaningful.

```
estimator.plot_macrostates(
    which="terminal",
    basis="umap",
    legend_loc="right",
    title="",
    size=100
)
```

12. Compute fate probabilities: to quantify cellular trajectories, assign each observation a fate probability toward the terminal states.

```
estimator.compute_fate_probabilities()
```

◆ **TROUBLESHOOTING**

13. (Optional) Plot fate probabilities: for each terminal state, the estimator method `plot_fate_probabilities` colors the observation according to the probability of this observation differentiating into that state (Extended Data Fig. 1b).

```
estimator.plot_fate_probabilities(
    same_plot=False, basis="umap", ncols=6
)
```

Alternatively, fate priming can be visualized through a circular projection: the corresponding method distributes the terminal states uniformly on a unit circle and arranges cells inside the circle to reflect the bias toward a specific fate (Fig. 3e).

```
cr.pl.circular_projection(
    adata, keys="celltype", legend_loc="right"
)
```

Stage 3: downstream analysis

14. (Optional) Identify lineage-correlated genes: to investigate why CellRank identified the MEP state, those genes whose expression correlate most with fate probability should be studied. Here, these genes are recovered for the MEP lineage (keyword `lineages`), restricted to stem cells (HSC) and MEPs (keyword `clusters`). Optionally, these genes can be plotted.

```
drivers = estimator.compute_lineage_drivers(
    return_drivers=True,
    cluster_key="celltype",
    lineages=["MEP"],
    clusters=["HSC", "MEP"],
)
estimator.plot_lineage_drivers(
    lineage="MEP",
    n_genes=10,
    ncols=5,
    title_fmt="{gene} corr={corr:.2}"
)
```

CellRank considers all genes, but according to best practices for single-cell analysis, cell-cycling genes and mitochondrial genes should not be considered among others,

for example[52]. These can be removed and the gene ranking visualized similar to how differentially expressed genes are commonly plotted (Fig. 3f).

```
# List of cell cycle-related genes
from crp.core import G2M_GENES, S_GENES
gene_prefixes = ("MT.", "RPL", "RPS", "^HB[^(p)]")
var_mask = (
    drivers.index.str.startswith(gene_prefixes)
    | drivers.index.isin(S_GENES + G2M_GENES)
)

gene_names = drivers.loc[~var_mask, :].index

ranking = pd.DataFrame(drivers.loc[gene_names, "MEP_corr"])
ranking["ranking"] = np.arange(len(gene_names))

df = ranking.iloc[:20, :]

fig, ax = plt.subplots(figsize=(6, 4))
y_min = np.min(df["MEP_corr"])
y_max = np.max(df["MEP_corr"])

y_min -= 0.1 * (y_max - y_min)
y_max += 0.4 * (y_max - y_min)
ax.set_ylim(y_min, y_max)
ax.set_xlim(-0.75, 19.5)

for gene in df.index:
    ax.text(
        df.loc[gene, "ranking"],
        df.loc[gene, "MEP_corr"],
        gene,
        rotation="vertical",
        verticalalignment="bottom",
        horizontalalignment="center",
        fontsize=20,
        color="#000000",
    )
```

15. (Optional) Fit and visualize GEX trends: the list of putative MEP lineage drivers includes known drivers of the megakaryocyte lineage among the top-ranked genes[54–59]. To study their GEX pattern, fit a Gaussian additive model to describe the lineage-specific change in expression over pseudotime (Fig. 3g).

```
model = cr.models.GAM(adata)

cr.pl.gene_trends(
    adata,
    model=model,
    genes=["ITGA2B", "VWF", "PLEK", "RGS18", "PIEZO2"],
    time_key="palantir_pseudotime",
    hide_cells=True,
    same_plot=True,
)
```

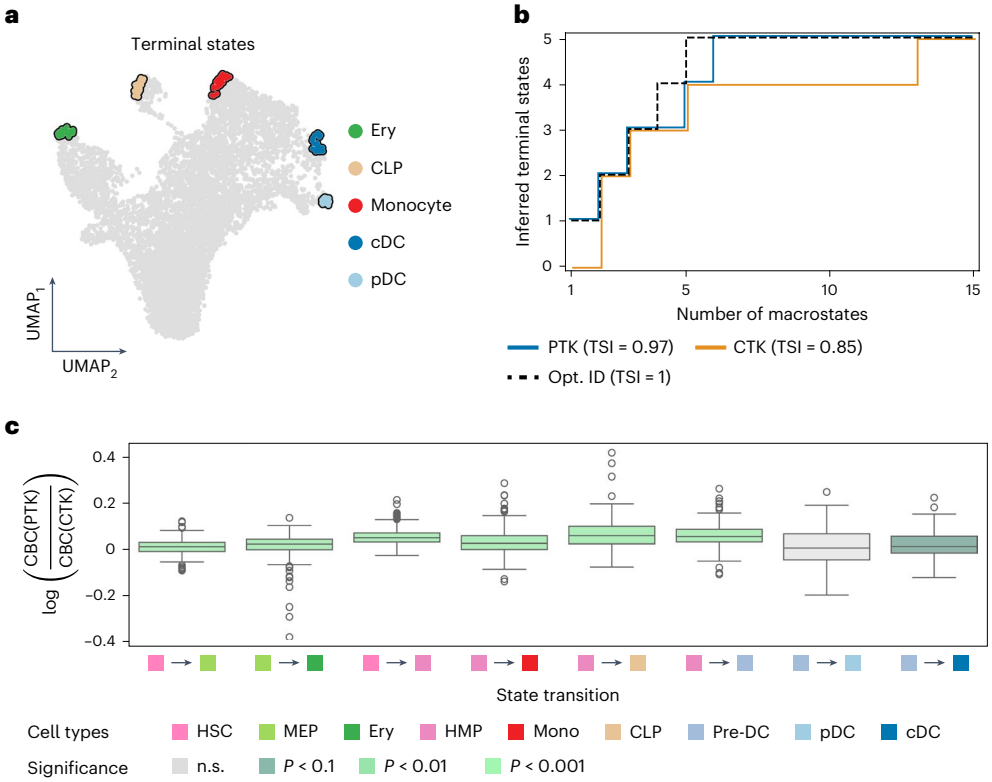


Fig. 4 | Comparing the performance of the PseudotimeKernel and CytoTRACEKernel. a, The UMAP embedding colored by the manually set terminal states. The color coding is identical to Fig. 2a, and the cells not associated with a macrostate are colored gray. **b**, The TSI curve for the PseudotimeKernel (solid blue), the CytoTRACEKernel (solid orange) and an optimal identification (dashed black). **c**, The log odds ratio of the CBC score using the PseudotimeKernel and CytoTRACEKernel. Box plots indicate the median (center line), interquartile range (hinges) and whiskers at 1.5× interquartile range (from left to right: 231, 190, 937, 375, 162, 238, 45, 49 cells). PTK, PseudotimeKernel; CTK, CytoTRACEKernel; Opt. ID, optimal identification; pDC, plasmacytoid dendritic cells; cDC, classical dendritic cells; CLP, common lymphoid progenitors; HSC, hematopoietic stem cells; HMP, hematopoietic multipotent progenitor cells; pre-DC, pre-dendritic cells; Ery, erythroid.

16. (Optional) Compute and compare TSI score: to compare the findings of this procedure with Procedure 1, set the terminal states equal to the expected, which are also used for the CytoTRACE-based analysis of the data (Fig. 4a); compute the TSI score next and visualize it compared with the previous analysis and an optimal identification (Fig. 4b).

```
estimator.set_terminal_states(TERMINAL_STATES)
tsi_score = estimator.tsi(
    n_macrostates=15,
    terminal_states=TERMINAL_STATES,
    cluster_key="celltype"
)

palette = {
    "PseudotimeKernel": "#DE8F05",
    "CytoTRACEKernel": "#DE8F05",
    "Optimal identification": "#000000"
}

with mplscience.style_context():
    sns.set_style(style="whitegrid")
```

```
    estimator.plot_tsi(palette=palette)
    plt.show()
```

17. (Optional) Compute CBC: the CBC is computed according to the a priori knowledge of cell state transitions. This requires computing fate probabilities and optionally plotting them (Extended Data Fig. 1c).

```
estimator.compute_fate_probabilities()
estimator.plot_fate_probabilities(
    same_plot=False, basis="umap", ncols=5
)

cluster_key="celltype"
rep="X_pca"

# Collect CBCs for each cell state transition in a Pandas DataFrame,
where
# each row corresponds to the CBC of a cell in the source population
with cells
# from the target state in its neighborhood
ptk_cbc = []
for source, target in tqdm(STATE_TRANSITIONS):
    _cbc = ptk.cbc(
        source=source,
        target=target,
        cluster_key=cluster_key,
        rep=rep
  )

  state_transition = [f"{source} - {target}"] * len(_cbc)
    ptk_cbc.append(
        pd.DataFrame(
            {
              "state_transition": state_transition,
              "cbc": _cbc,
          }
        )
    )
ptk_cbc = pd.concat(ptk_cbc)
```

18. (Optional) Compare kernel performance: to compare the PseudotimeKernel with the CytoTRACEKernel, load the CBC of the former, compute their log ratio and test if the mean ratio is significantly larger than 0, that is, if the PseudotimeKernel yielded significantly better results than the CytoTRACE-based analysis on the basis of a one-sided Welch's *t*-test. Plotting the log ratios summarized by boxplots and coloring each box by the significance of the test (Fig. 4c) and the comparison in general is again optional.

```
ctk_cbc = pd.read_parquet(
    DATA_DIR / "bone_marrow" / "results" / "ct_cbc.parquet"
)
cbc = pd.DataFrame(
  {
    "Log ratio": np.log((ptk_cbc["cbc"] + 1) / (ctk_cbc["cbc"] + 1)),
    "State transition": ptk_cbc["state_transition"]
```

```
    }
)

ttest_res, significances = get_ttest_res(cbc)
palette = get_palette(significances=significances)

plot_cbc_ratio(cbc=cbc, palette=palette, figsize=(12, 6))
```

Procedure 3: velocity-based fate mapping

▲ **CRITICAL** As an example of a velocity-based CellRank workflow, we feed a scRNA-seq spermatogenesis[45] dataset to our canonical pipeline. We present how to set up the kernel and combine it with a ConnectivityKernel; the subsequent steps shown in Procedures 1 and 2 work can then be applied here in an equivalent manner.

Read data

1. Load processed GEX data into the AnnData[53] format.

```
adata = ad.io.read_h5ad(DATA_DIR / "spermatogenesis" / "processed" /
"adata.h5ad")
```

Stage 1: estimate cell–cell transition probabilities

2. Initialize kernel and compute transition probabilities: the VelocityKernel takes the AnnData object with cell velocities and states as input to compute cell–cell transition probabilities.

```
vk = cr.kernels.VelocityKernel(adata)
```

3. Compute transition probabilities.

```
vk.compute_transition_matrix()
```

4. (Optional) Combine kernel dynamics: to take advantage of complementary views, CellRank can combine different kernels through a weighted sum of transition probabilities. Here, the VelocityKernel is combined with a ConnectivityKernel. As the velocity information is more sophisticated than transcriptomic similarity, the kernels are weighted in a 4:1 ratio (Fig. 5a); the previous procedures highlighted how to evaluate kernel performance consistently (Procedure 1, Steps 10 and 15 and Procedure 2, Steps 16–18).

```
ck = cr.kernels.ConnectivityKernel(adata).compute_transition_matrix()
combined_kernel = 0.8 * vk + 0.2 * ck
```

Stage 2: infer terminal states

5. Infer terminal states as shown in the examples in the preceding Procedures 1 and 2.

Stage 3: downstream analysis

6. Perform downstream analysis as shown in Procedures 1 and 2.

Procedure 4: OT-based fate mapping

▲ **CRITICAL** To showcase the incorporation of experimental time points, we apply our RealTimeKernel to a time-resolved hematopoiesis dataset[44] (Fig. 5b). We present how to set

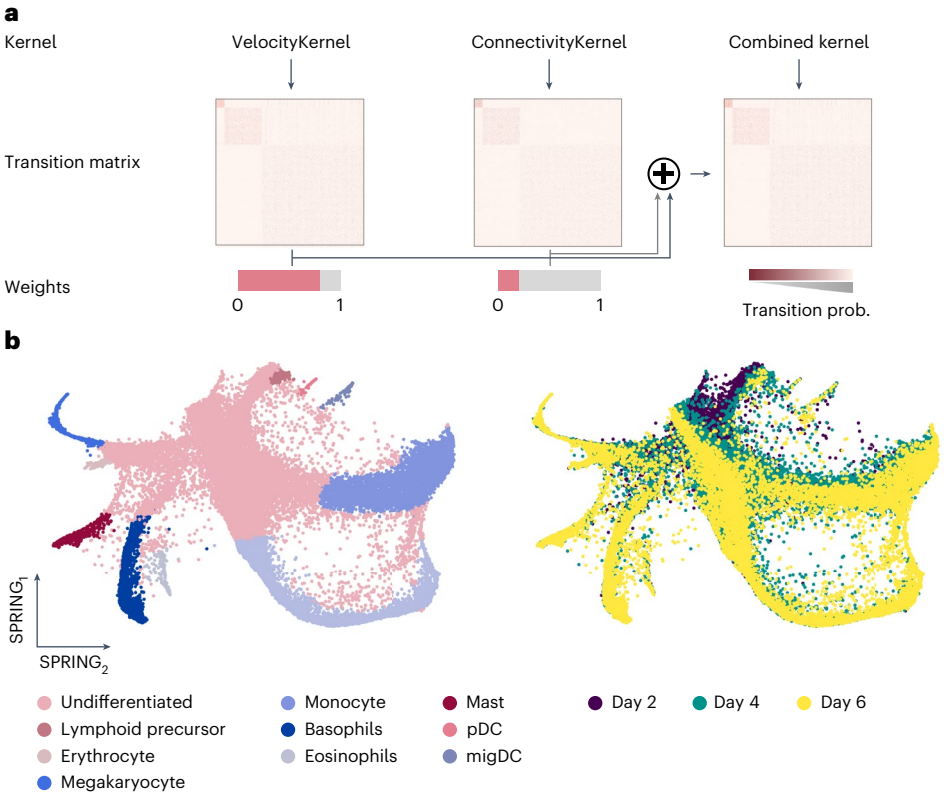


Fig. 5 | Kernel combination and application to time-resolved single-cell data. a, A schematic of how CellRank combines kernels through a weighted sum of their transition matrices. **b**, The SPRING[62] embedding of 32,721 human hematopoiesis cells, colored by their cell type (left) and experimental time point (right), according to the original publication[44]. pDC, plasmocytoid dendritic cell; migDC, Ccr7^{+} migratory dendritic cell.

up the kernel; the subsequent steps shown in Procedures 1 and 2 can then be applied here in an equivalent manner.

Read and prepare data

1. Load processed GEX data into the AnnData[53] format.

```
adata = ad.io.read_zarr(DATA_DIR / "larry" / "processed" / "adata.
zarr.zip")
```

Stage 1: estimate cell–cell transition probabilities

2. Compute OT couplings: here, we compute the OT couplings using moscot[25]. However, any OT framework can be used, as we and others have exemplified in the past by using WOT[15] and stationary OT[60], for example.

```
from moscot.problems.time import TemporalProblem

# Set the time point variable to categorical
adata.obs["day"] = adata.obs["day"].astype("category")

# Setup the OT problem
tp = TemporalProblem(adata)
tp = tp.prepare(time_key="day")
```

```
# Solve the OT problem
tp = tp.solve(epsilon=1e-3, tau_a=0.95, scale_cost="mean")
```

3. Initialize kernel: we initialize the RealTimeKernel using our moscot-computed couplings. If another framework is used, the kernel is initialized by passing the couplings directly.

```
rtk = cr.kernels.RealTimeKernel.from_moscot(tp)
```

4. Compute transition matrix: the transition matrix can include intra-time point dynamics characterized by a time-point-specific similarity graph; here, we consider these dynamics at each time point and assign them 0.2 weight compared with inter-time point connections.

```
rtk.compute_transition_matrix(self_transitions="all", conn_weight=0.2)
```

Stage 2: infer terminal states

5. Infer terminal states as shown in Procedure 1 and 2.

Stage 3: downstream analysis

6. Perform downstream analysis as shown in Procedure 1 and 2.

Troubleshooting

CellRank is a computational tool that depends on other Python libraries. As such, applying it to different data may result in numerical issues or software errors. As CellRank is freely available, users can open an issue at GitHub via https://github.com/theislab/cellrank/issues, describing the corresponding problem they encountered. Table 1 lists common problems we and others have previously encountered, together with possible solutions.

Table 1 | Troubleshooting table

Step	Problem	Possible reason	Solution
9 (Procedure 1 and Procedure 2)	Macrostate computation fails	GPCCA encountered a numerical issue	Switch to a different number of macrostates
			Use different parameters to compute the transition matrix if applicable
			Redefine the transition matrix by adding the ConnectivityKernel using a small weight; this approach has proven to improve the condition number of the matrix
13 (Procedure 1), 12 (Procedure 2)	Fate probability computation fails	GPCCA encountered a numerical issue	Use different parameters to compute the transition matrix if applicable
			Decrease the convergence tolerance for the default iterative solver
			Change to a direct solver with the keyword argument `solver="direct"`
			Redefine the transition matrix by adding the ConnectivityKernel using a small weight; this approach has proven to improve the condition number of the matrix

Timing

Procedure 1

Steps 1–3, data loading and preparation: 0.4 s
Steps 4–15, estimating fate probabilities and computing the CBC score: 27.2 s
Steps 4 and 5, estimating cell–cell transition probabilities: 0.7 s
Step 6–8, initializing estimators and compute Schur decomposition: 3.4 s

Steps 9–12, inferring terminal states: 22.5 s
Steps 13 and 14, estimating fate probabilities: 0.1 s
Step 15, computing CBC: 0.5 s

Procedure 2
Steps 1–2, data loading and preparation: 0.5 s
Steps 3–18, estimating fate probabilities and performing downstream analyses: 9.2 s
Steps 3–5, estimating cell–cell transition probabilities: 0.7 s
Steps 6–8, initializing estimators and compute Schur decomposition: 3.2 s
Steps 9–11, inferring terminal states: 0.6 s
Steps 12 and 13, estimating fate probabilities: 0.5 s
Step 14, identifying lineage-correlated genes: 1.2 s
Step 15, fitting GEX patterns: 1.9 s
Step 16–18, comparing kernel performance: 1.1 s

Procedure 3
Step 1, data loading and preparation: 0.05 s
Steps 2–4, estimating cell–cell transition probabilities: 3.0 s
Steps 2 and 3, estimating VelocityKernel-based transition matrix: 3.0 s
Step 4, combining kernels: 0.006 s

Procedure 4
Step 1, data loading and preparation: 12.7 s
Steps 2–4, estimating cell–cell transition probabilities: 43.5 s
Step 2, computing OT couplings: 39.4 s
Steps 3 and 4, initializing RealTimeKernel and computing transition matrix: 4.1 s

Anticipated results

CellRank is versatile in its application and, thus, its outputs may vary between different analyses. Nonetheless, these outputs generally fall into three categories: (1) outputs observed by any full analysis workflow, (2) outputs to assess and compare model performance and (3) outputs to probe model behavior and generate hypotheses.

Essential outputs common to all CellRank-based trajectory inference pipelines are cell–cell transition probabilities, resulting terminal states and cell-specific fate probabilities toward them. These characteristics form the basis for any subsequent analysis. As the generated outputs are high-dimensional, CellRank provides different visualization concepts: coloring inferred terminal cells in a low-dimensional data representation such as UMAP or *t*-distributed stochastic neighbor embeddings illustrates their cell state intuitively (Figs. 2e, 3d and 4a); coloring each cell in the same representation by its fate probability toward each of these states conveys putative lineage priming (Extended Data Fig. 1); and circular projections of cells according to their fate bias toward terminal states provides the same information in a different format (Fig. 3e).

Model performance-related outputs comprise the TSI and CBC scores. The TSI score quantifies how well a given estimator–kernel pair identifies known terminal states in absolute terms (Fig. 2d; Procedure 1, Step 10 and Procedure 2, Step 16) but also relative to another setup (Fig. 4b; Procedure 2, Step 16). In the ideal case, a kernel recovers a new terminal state for every additional macrostate until all terminal states have been identified. The corresponding TSI plot is a canonical step function that remains constant once the number of macrostates matches the number of terminal states. Consequently, kernels reaching this constant level earlier recapitulate the underlying biology better and are thus preferred. To quantify this notion, CellRank computes the ratio of the area under a kernel's TSI curve and the optimal curve—the TSI score (Figs. 2d and 4b). Relatedly, each kernel provides a CBC score, but interpreting the

measure directly may prove difficult owing to the lack of a reference baseline; comparing the log ratio of two kernels, however, provides such a reference (Fig. 4c). Here, positive log ratios correspond to cell transitions adhering more faithfully to ground truth under the PseudotimeKernel compared with the CytoTRACEKernel and vice versa for negative values; Welch's *t*-tests allow quantifying if one kernel results in significantly better recapitulation of known cell state changes.

Downstream analyses can generate a multitude of results: ranking genes according to their correlation between GEX and fate probability toward a given terminal state suggests putative lineage drivers (Fig. 3f). Similarly, fitting generalized additive models to weighted GEX over pseudotime visualizes lineage-specific gene trends (Fig. 3g) and clustering them identifies sets of genes with similar expression trends (`cellrank.pl.cluster_trends`). Alternatively, to compare GEX trends and propose putative causal relationships between genes, CellRank allows ranking a set of genes according to their peak in pseudotime and visualizing the expression pattern via a heat map (`cellrank.pl.heatmap`); each row corresponds to a gene, ordered from early to late peaking, and each column to the corresponding GEX at a given pseudotime value. Taken together, these downstream analyses link fate probabilities to GEX change and provide means for hypothesis generatation.

Data availability

All datasets used for the presented procedures are publicly available through the original studies. For convenience, the raw and processed count matrices are available via Figshare at https://doi.org/10.6084/m9.figshare.c.7752290.v1 (ref. 41).

Code availability

CellRank is publicly available under a BSD-3-Clause license, with code available via GitHub at https://github.com/theislab/cellrank and deposited via Zenodo at https://doi.org/10.5281/zenodo.10210196 (ref. 61). The Python code to run the presented analyses is available via GitHub at https://github.com/theislab/cellrank_protocol. The code in this Protocol has been peer reviewed.

Received: 10 October 2024; Accepted: 24 November 2025;
Published online: 29 January 2026

References

1. Kanemaru, K. et al. Spatially resolved multiomics of human cardiac niches. *Nature* **619**, 801–810 (2023).
2. Suo, C. et al. Mapping the developing human immune system across organs. *Science* **376**, eabo0510 (2022).
3. Siletti, K. et al. Transcriptomic diversity of cell types across the adult human brain. *Science* **382**, eadd7046 (2023).
4. Regev, A. et al. Science forum: the human cell atlas. *eLife* https://doi.org/10.7554/eLife.27041 (2017).
5. Haniffa, M. et al. A roadmap for the Human Developmental Cell Atlas. *Nature* **597**, 196–205 (2021).
6. Sikkema, L. et al. An integrated cell atlas of the lung in health and disease. *Nature Medicine* **29**, 1563–1577 (2023).
7. Yayon, N. et al. A spatial human thymus cell atlas mapped to a continuous tissue axis. *Nature* **635**, 708–718 (2024).
8. Li, H. et al. Fly cell atlas: a single-nucleus transcriptomic atlas of the adult fruit fly. *Science* https://doi.org/10.1126/science.abk2432 (2022).
9. Yao, Z. et al. A high-resolution transcriptomic and spatial atlas of cell types in the whole mouse brain. *Nature* **624**, 317–332 (2023).
10. Braun, E. et al. Comprehensive cell atlas of the first-trimester developing human brain. *Science* https://doi.org/10.1126/science.adf1226 (2023).
11. Fleck, J. S. et al. Inferring and perturbing cell fate regulomes in human brain organoids. *Nature* **621**, 365–372 (2022).
12. Pijuan-Sala, B. et al. A single-cell molecular map of mouse gastrulation and early organogenesis. *Nature* **566**, 490–495 (2019).
13. Setty, M. et al. Characterization of cell fate probabilities in single-cell data with Palantir. *Nat. Biotechnol.* **37**, 451–460 (2019).
14. Lange, M. et al. CellRank for directed single-cell fate mapping. *Nat. Methods* **19**, 159–170 (2022).
15. Schiebinger, G. et al. Optimal-transport analysis of single-cell gene expression identifies developmental trajectories in reprogramming. *Cell* **176**, 928–943.e22 (2019.
16. Weiler, P., Lange, M., Klein, M., Pe'er, D. & Theis, F. CellRank 2: unified fate mapping in multiview single-cell data. *Nat. Methods* **21**, 1196–1205 (2024).
17. Xiao, Y. et al. Tracking single-cell evolution using clock-like chromatin accessibility loci. *Nat. Biotechnol.* **43**, 784–798 (2025).
18. Meier, A. B. et al. Epicardioid single-cell genomics uncovers principles of human epicardium biology in heart development and disease. *Nat. Biotechnol.* **41**, 1787–1800 (2023).
19. Reuter, B., Fackeldey, K. & Weber, M. Generalized Markov modeling of nonreversible molecular kinetics. *J. Chem. Phys.* **150**, 174103 (2019).
20. Van den Berge, K. et al. Trajectory-based differential expression analysis for single-cell sequencing data. *Nat. Commun.* **11**, 1–13 (2020).
21. Mittnenzweig, M. et al. A single-embryo, single-cell time-resolved model for mouse gastrulation. *Cell* **184**, 2825–2842.e22 (2021).
22. Lee, D. et al. Plasticity of human microglia and brain perivascular macrophages in aging and Alzheimer's disease. Preprint at *medRxiv* https://doi.org/10.1101/2023.10.25.23297558 (2024).
23. Remsik, J. et al. Interferon-γ orchestrates leptomeningeal anti-tumour response. *Nature* **643**, 1087–1096 (2025).
24. Klein, D. Mapping cells through time and space with moscot. *Nature* **638**, 1065–1075 (2025).

25. Lange, M. et al. Mapping lineage-traced cells across time points with moslin. *Genome Biol.* **25**, 1–38 (2024).
26. Street, K. et al. Slingshot: cell lineage and pseudotime inference for single-cell transcriptomics. *BMC Genomics* **19**, 1–16 (2018).
27. Lönnberg, T. et al. Single-cell RNA-seq and computational analysis using temporal mixture modeling resolves TH1/TFH fate bifurcation in malaria. *Sci. Immunol.* https://doi.org/10.1126/sciimmunol.aal2192 (2017).
28. Herman, J. S., Sagar & Grün, D. FateID infers cell fate bias in multipotent progenitors from single-cell RNA-seq data. *Nat. Methods* **15**, 379–386 (2018).
29. Velten, L. et al. Human haematopoietic stem cell lineage commitment is a continuous process. *Nat. Cell Biol.* **19**, 271–281 (2017).
30. La Manno, G. et al. RNA velocity of single cells. *Nature* **560**, 494–498 (2018).
31. Qiu, X. et al. Mapping transcriptomic vector fields of single cells. *Cell* **185**, 690–711.e45 (2022).
32. Bergen, V., Soldatov, R. A., Kharchenko, P. V. & Theis, F. J. RNA velocity-current challenges and future perspectives. *Mol. Syst. Biol.* **17**, e10282 (2021).
33. Soneson, C., Srivastava, A., Patro, R. & Stadler, M. B. Preprocessing choices affect RNA velocity results for droplet scRNA-seq data. *PLOS Comput. Biol.* **17**, e1008585 (2021).
34. Barile, M. et al. Coordinated changes in gene expression kinetics underlie both mouse and human erythroid maturation. *Genome Biol.* **22**, 1–22 (2021).
35. Bergen, V., Lange, M., Peidli, S., Wolf, F. A. & Theis, F. J. Generalizing RNA velocity to transient cell states through dynamical modeling. *Nat. Biotechnol.* **38**, 1408–1414 (2020).
36. Gayoso, A. et al. Deep generative modeling of transcriptional dynamics for RNA velocity analysis in single cells. *Nat. Methods* **21**, 50–59 (2024).
37. Moorman, A. R. et al. Progressive plasticity during colorectal cancer metastasis. *Nature* **637**, 947–954 (2025).
38. Weiler, P., Van den Berge, K., Street, K. & Tiberi, S. A guide to trajectory inference and RNA velocity. *Methods Mol. Biol.* **2584**, 269–292 (2023).
39. Qiu, X. et al. Reversed graph embedding resolves complex single-cell trajectories. *Nat. Methods* **14**, 979–982 (2017).
40. Haghverdi, L., Büttner, M., Wolf, F. A., Buettner, F. & Theis, F. J. Diffusion pseudotime robustly reconstructs lineage branching. *Nat. Methods* **13**, 845–848 (2016).
41. Weiler, P. & Theis, F. CellRank protocol. *Figshare* https://doi.org/10.6084/m9.figshare.c.7752290.v1 (2025).
42. Persad, S. et al. SEACells infers transcriptional and epigenomic cellular states from single-cell genomics data. *Nat. Biotechnol.* **41**, 1746–1757 (2023).
43. McInnes, L., Healy, J. & Melville, J. UMAP: uniform manifold approximation and projection for dimension reduction. (2018).
44. Weinreb, C., Rodriguez-Fraticelli, A., Camargo, F. D. & Klein, A. M. Lineage tracing on transcriptional landscapes links state to fate during differentiation. *Science* **367** (2020).
45. Hermann, B. P. et al. The mammalian spermatogenesis single-cell transcriptome, from spermatogonial stem cells to spermatids. *Cell Rep.* **25**, 1650–1667.e8 (2018).
46. Wang, W. et al. RegVelo: gene-regulatory-informed dynamics of single cells. Preprint at *bioRxiv* https://doi.org/10.1101/2024.12.11.627935 (2024).
47. Gao, M., Qiao, C. & Huang, Y. UniTVelo: temporally unified RNA velocity reinforces single-cell trajectory inference. *Nat. Commun.* **13**, 1–11 (2022).
48. Li, S. et al. A relay velocity model infers cell-dependent RNA velocity. *Nat. Biotechnol.* **42**, 99–108 (2023).
49. Li, J., Pan, X., Yuan, Y. & Shen, H.-B. TFvelo: gene regulation inspired RNA velocity estimation. *Nat. Commun.* **15**, 1–15 (2024).
50. Burdziak, C. et al. scKINETICS: inference of regulatory velocity with single-cell transcriptomics data. *Bioinformatics* **39**, i394–i403 (2023).
51. Battich, N. et al. Sequencing metabolically labeled transcripts in single cells reveals mRNA turnover strategies. *Science* **367**, 1151–1156 (2020).
52. Heumos, L. et al. Best practices for single-cell analysis across modalities. *Nat. Rev. Genet.* **24**, 550–572 (2023).
53. Virshup, I., Rybakov, S., Theis, F. J., Angerer, P. & Alexander Wolf, F. anndata: access and store annotated data matrices. *J. Open Source Softw.* **9**, 4371 (2024).
54. Dumon, S. et al. Itga2b regulation at the onset of definitive hematopoiesis and commitment to differentiation. *PLoS ONE* **7**, e43300 (2012).
55. Psaila, B. et al. Single-cell analyses reveal megakaryocyte-biased hematopoiesis in myelofibrosis and identify mutant clone-specific targets. *Mol. Cell* **78**, 477–492.e8 (2020).
56. Bigot, T. et al. Single-cell analysis of megakaryopoiesis in peripheral CD34$^+$ cells: insights into ETV6-related thrombocytopenia. *J. Thromb. Haemost.* **21**, 2528–2544 (2023).
57. Leiva, O. et al. The role of extracellular matrix stiffness in megakaryocyte and platelet development and function. *Am. J. Hematol.* **93**, 430–441 (2018).
58. Delesque-Touchard, N. et al. Regulator of G-protein signaling 18 controls both platelet generation and function. *PLoS ONE* **9**, e113215 (2014).
59. Yowe, D. et al. RGS18 is a myeloerythroid lineage-specific regulator of G-protein-signalling molecule highly expressed in megakaryocytes. *Biochem J.* **359**, 109–118 (2001).
60. Zhang, S., Afanassiev, A., Greenstreet, L., Matsumoto, T. & Schiebinger, G. Optimal transport analysis reveals trajectories in steady-state systems. *PLOS Comput. Biol.* **17**, e1009466 (2021).
61. theislab/cellrank: v2.0.2. *Zenodo* https://doi.org/10.5281/zenodo.10210197.
62. Weinreb, C., Wolock, S. & Klein, A. M. SPRING: a kinetic interface for visualizing high dimensional single-cell expression data. *Bioinformatics* **34**, 1246–1248 (2018).
63. Bendall, S. C. et al. Single-cell trajectory detection uncovers progression and regulatory coordination in human B cell development. *Cell* **157**, 714–725 (2014).
64. Tritschler, S. et al. Concepts and limitations for learning developmental trajectories from single cell genomics. *Development* **146** (2019).
65. Halmos, P. et al. DeST-OT: alignment of spatiotemporal transcriptomics data. *Cell Syst.* **16**, 101160 (2025).
66. Erhard, F. et al. Time-resolved single-cell RNA-seq using metabolic RNA labelling. *Nat. Rev. Methods Primers* **2**, 1–18 (2022).
67. Cao, J., Zhou, W., Steemers, F., Trapnell, C. & Shendure, J. Sci-fate characterizes the dynamics of gene expression in single cells. *Nat. Biotechnol.* **38**, 980–988 (2020).
68. Erhard, F. et al. scSLAM-seq reveals core features of transcription dynamics in single cells. *Nature* **571**, 419–423 (2019).
69. Qiu, Q. et al. Massively parallel and time-resolved RNA sequencing in single cells with scNT-seq. *Nat. Methods* **17**, 991–1001 (2020).

Acknowledgements
We thank M. Lange, M. Klein and D. Pe'er for their invaluable contributions and collaboration to develop CellRank.

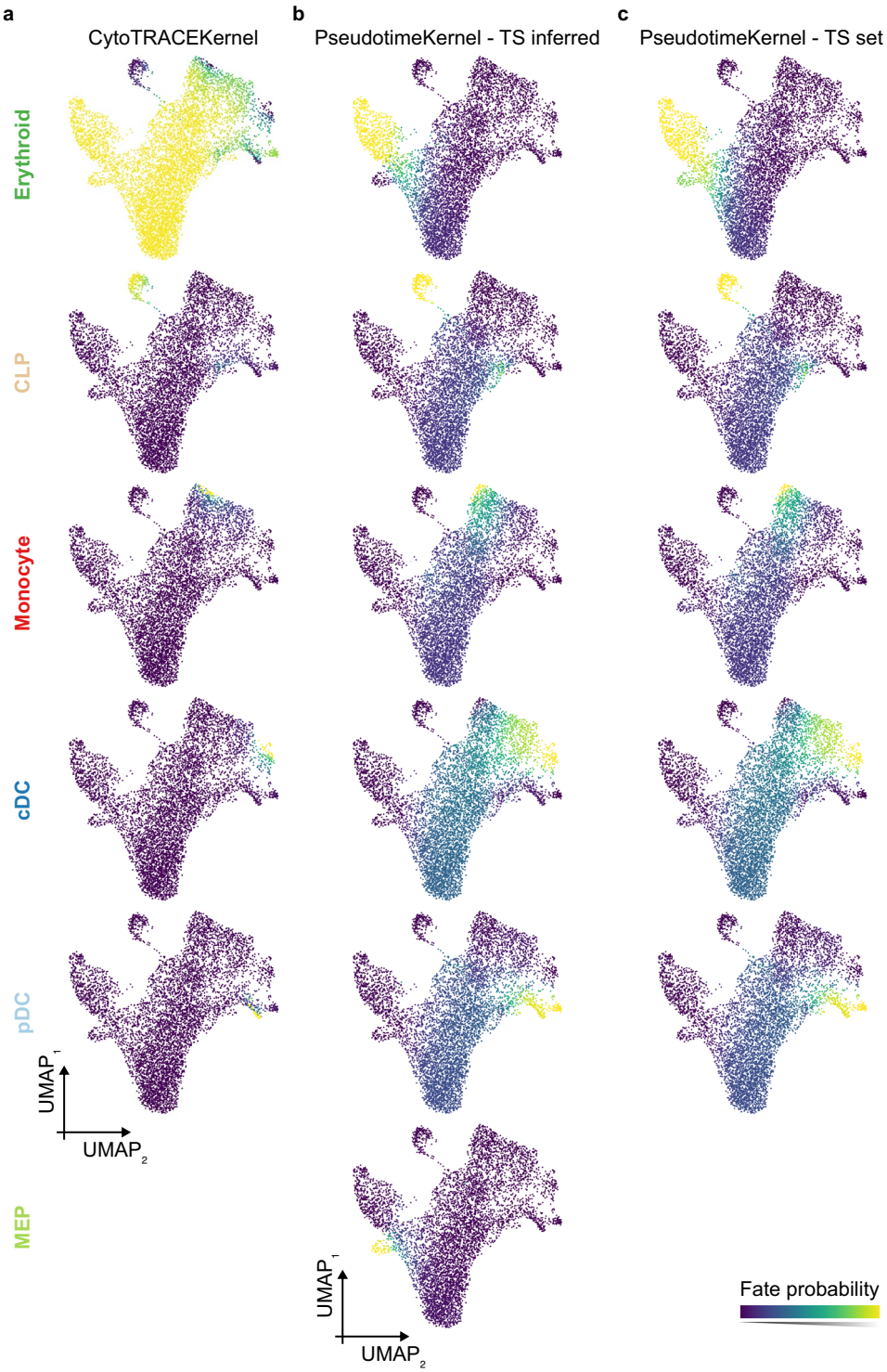
Author contributions
P.W. conceptualized the study and performed the analyses. P.W. and F.J.T. wrote the manuscript.

Competing interests
F.J.T. consults for Immunai, CytoReason, BioTuring and Genbio and Valinor Industries, and has ownership interest in RN.AI Therapeutics, Dermagnostix GmbH and Cellarity. The other authors declare no competing interests.

Additional information
Extended data is available for this paper at https://doi.org/10.1038/s41596-025-01314-w.

Correspondence and requests for materials should be addressed to Fabian J. Theis.

Peer review information *Nature Protocols* thanks Zhicheng Ji, Haotian Zhuang and the other, anonymous, reviewer(s) for their contribution to the peer review of this work.



Extended Data Fig. 1 | Fate probability from different approaches. a–**c**. UMAP embedding of human bone marrow data colored by fate probabilities inferred by the CytoTRACEKernel (**a**.) and PseudotimeKernel using automatically inferred (**b**.) and manually defined (**c**.) terminal states (TSs).