Bioinformatics, 2025, **41(12)**, btaf624
https://doi.org/10.1093/bioinformatics/btaf624
Advance Access Publication Date: 14 November 2025
Applications Note

OXFORD

Systems biology

FastSCODE: an accelerated SCODE algorithm for inferring gene regulatory networks on manycore processors

Rakbin Sung[1,†], Seongmi Woo[1,†], Dongmin Shin[1,†], Junil Kim[2,3,*], Daewon Lee[1,4,*]

[1]Department of Applied Art and Technology, College of Art and Technology, Chung-Ang University, Anseong 17546, Republic of Korea
[2]School of Systems Biomedical Science, Soongsil University, Seoul 06978, Republic of Korea
[3]Department of Bioinformatics, Soongsil University, Seoul 06978, Republic of Korea
[4]School of Art and Technology, College of Art and Technology, Chung-Ang University, Anseong 17546, Republic of Korea

*Corresponding authors. Daewon Lee, School of Art and Technology, College of Art and Technology, Chung-Ang University, 4726 Seodong-daero, Daedeok-myeon, Anseong 17546, Republic of Korea. E-mail: dwlee@cau.ac.kr; Junil Kim, School of Systems Biomedical Science, Soongsil University, 369 Sangdo-Ro, Dongjak-Gu, Seoul 06978, Republic of Korea. E-mail: junilkim@ssu.ac.kr

†= equal contribution.

Associate Editor: Jianlin Cheng

Abstract

Summary: SCODE reconstructs gene regulatory networks from single-cell RNA sequencing (scRNA-seq) data using an ordinary differential equation (ODE) model, and has been successfully applied to a wide range of scRNA-seq datasets, including mouse, human, and plant cells. However, its computational performance is limited when processing large datasets due to its sequential execution flow and repeated optimization loops. To overcome this limitation, we have developed FastSCODE, a batch computing version of the SCODE algorithm optimized for acceleration on manycore processors such as GPUs. FastSCODE performs batch computation on multiple gene expression profiles and optimizes the parameters of a linear ODE model using manycore computing. Compared to the original implementation, FastSCODE achieves up to 6000× improvement in performance (from about one month to 10 min) on the CeNGEN scRNA-seq dataset when using multiple GPUs.

Availability and implementation: FastSCODE is publicly available on GitHub at https://github.com/cxinsys/fastscode.

1 Introduction

The expression values of genes can be tracked through RNA sequencing. The process of bulk RNA sequencing involves the calculation of the mean expression value of a given gene across multiple cells. This results in low resolution for expression dynamics from each individual cell. Single-cell RNA sequencing (scRNA-seq) acquires gene expressions from individual cells, with the objective of expanding our understanding of cellular dynamics. Applying computational algorithms to scRNA-seq data enables the inference of gene regulatory networks (GRNs), providing a profound understanding of the intricate mechanisms underlying complex biological phenomena (Pratapa *et al.* 2020, Kim *et al.* 2024). Consequently, the advent of scRNA-seq has led to the accumulation of large-scale datasets, thereby increasing the demand for highly efficient and scalable algorithms and software for big data analysis (Eisenstein 2020, Andrews *et al.* 2021, Granja *et al.* 2021). For example, GRNBoost2 is an accelerated version of GENIE3 that utilizes parallel processing with multiple CPUs (Moerman *et al.* 2019), while FastTENET achieves up to a 973-fold speedup over the original TENET by using GPU-based manycore computing (Sung *et al.* 2024).

SCODE is a GRN inference algorithm that models gene expression dynamics using a linear ODE framework and estimates regulatory influences through linear regression optimization (Matsumoto *et al.* 2017). The resulting score matrix quantitatively represents the strength of regulatory relationships between transcription factors and their target genes. SCODE has demonstrated its effectiveness in identifying key regulators through GRN inference across a variety of datasets, including mouse embryonal carcinoma (EC) cells, mouse embryonic fibroblasts (MEFs), and human EC cells (Matsumoto *et al.* 2017). Its application has extended to reconstructing GRNs from sequencing data obtained from plant tissues such as root, pericarp, and stem cells (Denyer *et al.* 2019, Zhang *et al.* 2021, Li *et al.* 2024), as well as from immune cell populations (Yossef *et al.* 2023).

As a representative algorithm based on a linear ODE model, SCODE is often used as a benchmark in comparative evaluations of newly developed GRN inference methods (Pratapa *et al.* 2020, Ramachandran *et al.* 2020, Shu *et al.* 2021, Su *et al.* 2022, Wang *et al.* 2022, Zinati *et al.* 2024, Chen *et al.* 2025). However, its original implementation suffers from limited computational efficiency, particularly when applied to large-scale datasets. The sequential processing of gene expression profiles and the iterative nature of ODE-based optimization, which involves multiple nested loops, lead to substantial performance degradation as the number of genes increases. To overcome the computational limitations of the original SCODE implementation, we have developed "FastSCODE," which is accelerated through parallel processing on manycore processors. FastSCODE reduces the number

Received: 21 July 2025; **Revised:** 28 October 2025; **Accepted:** 8 November 2025

of optimization loops by performing batch computation across multiple gene expression profiles and corresponding parameter vectors of the linear ODE model. It is specifically designed to support high performance manycore architectures such as GPUs, TPUs, and NPUs, which are well suited for large-scale batch operations. In FastSCODE, users can select an acceleration framework for manycore computing, such as NumPy (Harris *et al.* 2020), PyTorch (Paszke *et al.* 2019), CuPy (Okuta *et al.* 2017), TensorFlow (Abadi *et al.* 2015), or JAX (Bradbury *et al.* 2018), all of which are supported through a unified array computing interface. As a result, FastSCODE achieved more than an 800-fold speedup compared to the original SCODE implementation when executed on four NVIDIA RTX 4090 GPUs.

2 Methods

2.1 Basic concept

The SCODE algorithm quantifies regulatory relationships among genes by computing a score matrix from gene expression data. The observed gene expression data can be represented as $\mathbf{X} \in \mathbb{R}^{G \times C}$, where G and C denote the number of genes and cells, respectively. Equation (1) shows how SCODE models gene expression dynamics with a linear ODE model.

$$\frac{d\mathbf{x}}{dt} = \mathbf{A}\mathbf{x}, \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^{G}$ is a vector of the gene expression at the time t, and $\mathbf{A} \in \mathbb{R}^{G \times G}$ denotes the score matrix that indicates the strength of the relationship between genes in GRNs. However, the direct optimization of $\mathbf{A}$ has a time complexity of $O(CG^3)$, leading to computationally expensive for large-scale datasets. For reducing the computational complexity, SCODE introduces an additional linear ODE equation with a low-dimensional latent vector $\mathbf{z}$, as shown in Equation (2).

$$\begin{aligned} d\mathbf{z} &= \mathbf{B}\mathbf{z}dt, \\ \mathbf{x} &= \mathbf{W}\mathbf{z}, \end{aligned} \tag{2}$$

where $\mathbf{z} \in \mathbb{R}^{D}$ $(D \ll G)$ represents the latent vector that captures the underlying gene expression dynamics, and $\mathbf{B} \in \mathbb{R}^{D \times D}$ defines the linear dynamics of $\mathbf{z}$ in the latent space. The matrix $\mathbf{W} \in \mathbb{R}^{G \times D}$ is a linear mapping from the latent space to the gene expression vector. Given that $\mathbf{W}$ satisfies $\mathbf{x} = \mathbf{W}\mathbf{z}$, the ODE of $\mathbf{x}$ can be derived according to Equation (3).

$$d\mathbf{x} \approx \mathbf{W}\mathbf{B}\mathbf{W}^{+}\mathbf{x}dt, \tag{3}$$

where $\mathbf{W}^{+}$ denotes the Moore-Penrose pseudoinverse of $\mathbf{W}$. The score matrix $\mathbf{A}$ can be approximated by $\mathbf{A} \approx \mathbf{W}\mathbf{B}\mathbf{W}^{+}$ according to Equations (1) and (3). SCODE optimizes the matrix $\mathbf{B}$ through Monte Carlo sampling. In the first step of optimization, the diagonal elements of $\mathbf{B}$, denoted by the vector $\mathbf{b} \in \mathbb{R}^{D}$, are initialized with values uniformly sampled from the range $[b_{\min}, b_{\max}]$. Following the sampling of $\mathbf{b}$, the latent representation matrix $\mathbf{Z}$ is constructed by computing the general solution $\mathbf{Z}_{i,c} = e^{b_i t_c}$, where b_i is the i-th diagonal element of $\mathbf{B}$ and t_c denotes the c-th pseudotime point. Subsequently, $\mathbf{W}$ is derived from $\mathbf{Z}$ according to $\mathbf{W}^{\top} = (\mathbf{Z}\mathbf{Z}^{\top})^{-1}\mathbf{Z}\mathbf{X}^{\top}$ based on the relationship $\mathbf{X}^{\top} = \mathbf{Z}^{\top}\mathbf{W}^{\top}$. SCODE calculates the residual sum of squares (RSS) between $\mathbf{X}$ and $\mathbf{W}\mathbf{Z}$. During the optimization loop, one element of $\mathbf{b}$ is randomly sampled and updated in each iteration to minimize the RSS. Finally, the score matrix $\mathbf{A}$ is estimated as $\mathbf{A} \approx \mathbf{W}\mathbf{B}_{best}\mathbf{W}^{+}$, where $\mathbf{W}$ is computed using the matrix $\mathbf{B}_{best}$ that achieves the minimum RSS.

Although the original implementation of SCODE incorporates several techniques to mitigate computational complexity, it still requires a substantial amount of execution time due to independent computations for each gene and repeated optimization steps. It is evident that the execution time of SCODE increases dramatically with the number of genes and optimization iterations, making it computationally prohibitive for large-scale datasets.

2.2 Batch computing for reducing repetition

FastSCODE introduces batch array computing when solving linear regression to minimize the repeated computations for each gene. A subset of gene expressions is uploaded to a manycore processor, according to batch size B of FastSCODE. Depending on the chunk size B_c, FastSCODE computes the linear transformation on multiple gene expressions (Fig. 1A). If sufficient computational resources are available, computations across all genes can be executed in a single batch, eliminating the memory transfer overhead between the CPU and manycore processor. Furthermore, FastSCODE expands $\mathbf{B}$ into a $B_S \times D$ matrix for the parallel computation of B_S RSS values (Fig. 1A), significantly reducing the number of required optimization iterations while preserving the original random sampling strategy.

2.3 Parallel processing on manycore processors

In FastSCODE, scRNA-seq datasets are partitioned into batches according to the available manycore processors. Multiple worker processes are launched, each assigned to a specific manycore processor to perform linear transformations and RSS computations on designated data chunks. Users can configure the list of accessible manycore processors and specify the batch and chunk size for each processing unit. After computation, the RSS results from all batches are aggregated into a single array in CPU memory managed by the main process (Fig. 1A). Numerical operations involving large arrays can benefit from substantial speedup when executed on specialized manycore processors such as GPUs, NPUs, and TPUs. FastSCODE supports a variety of acceleration frameworks such as CuPy, JAX, TensorFlow, and PyTorch.

2.4 Experimental setup

To evaluate the performance of FastSCODE in comparison to SCODE, we conducted a series of experiments using four scRNA-seq datasets. The mouse embryonic stem cell (mESC) neural differentiation (Setty *et al.* 2016, Tuck *et al.* 2018) and the skin cancer (Gaydosik *et al.* 2020, Ji *et al.* 2020, Kfoury *et al.* 2021) datasets were selected, as they consist of fewer than several thousand genes and cells. Furthermore, two large-scale datasets were used: the zebrafish embryonic (Wagner *et al.* 2018) and the complete gene expression map of an entire nervous system (CeNGEN) (Hammarlund *et al.* 2018) datasets.

We evaluated the performance of FastSCODE across seven distinct computing systems (Table 1, available as supplementary data at *Bioinformatics* online). The execution time of SCODE on System 1 was used as the reference baseline for all comparisons. FastSCODE was tested using up to 4 GPUs, while for Systems 6 and 7, it utilized up to 2 GPUs, depending

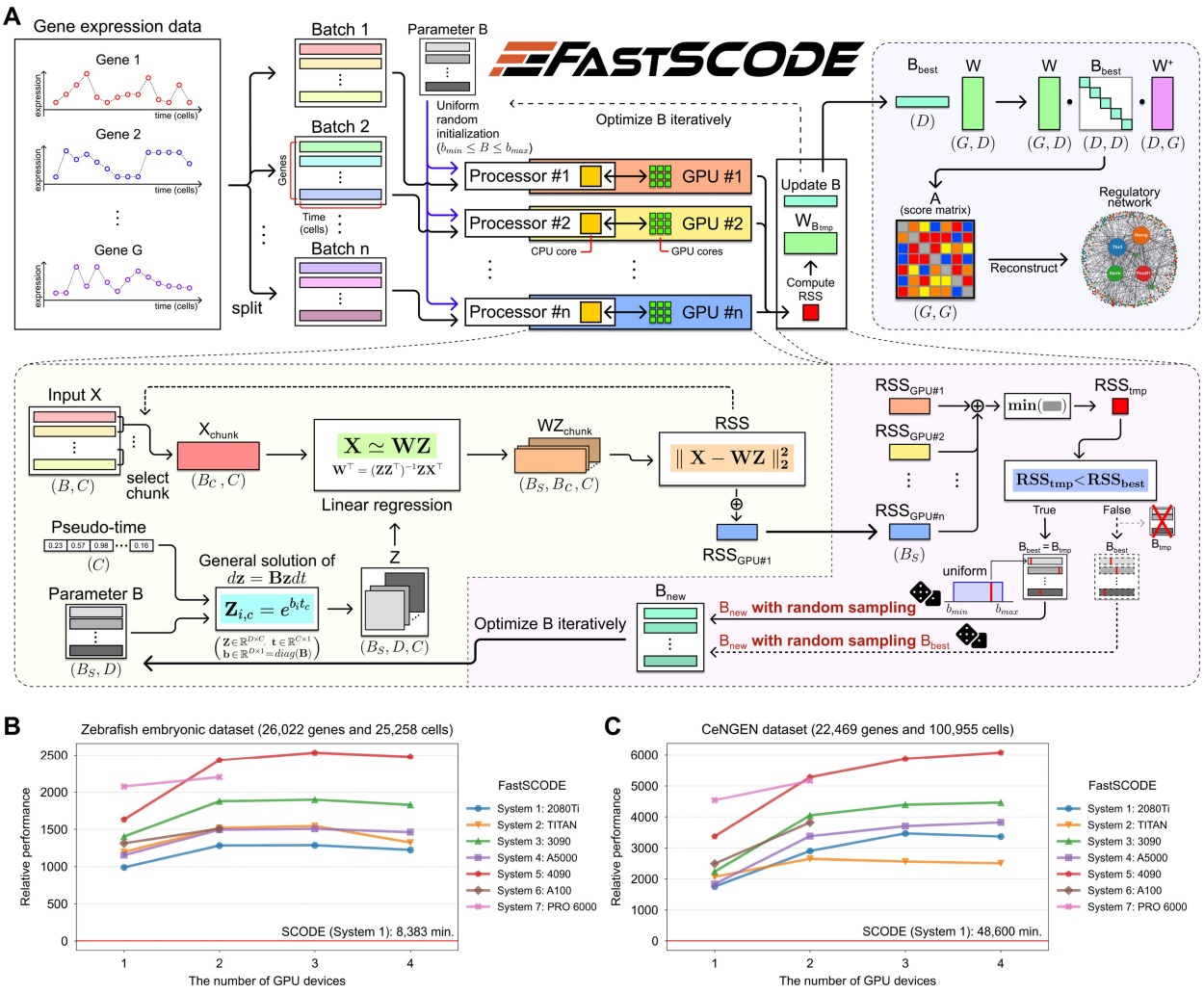

Figure 1. Overview of FastSCODE and performance analysis. (A) Schematic overview of FastSCODE algorithm. (B and C) Relative performance of FastSCODE on the (B) zebrafish embryonic dataset, and (C) CeNGEN dataset across multiple computing systems. All baseline experiments of the original SCODE were conducted using 32 CPU cores of Intel Xeon Silver 4214R on System 1. Refer to Table 1, available as supplementary data at *Bioinformatics* online for details on system configurations.

on hardware configuration of the system. The batch size was set as the number of genes divided by the number of GPUs, while the chunk size was dynamically adjusted by the algorithm. Relative performance was calculated by dividing the execution time of SCODE by that of FastSCODE. Additionally, we analyzed the runtime breakdown by measuring the resource utilization of FastSCODE and SCODE.

3 Results

3.1 Computation speed improvement

On the zebrafish dataset, FastSCODE achieved up to a 2532-fold speedup when utilizing three NVIDIA RTX 4090 GPUs (Fig. 1B). Even greater acceleration was observed for the CeNGEN dataset, where FastSCODE achieved >6000-fold speedup using four 4090 GPUs (Fig. 1C). The actual execution time was reduced from 8383 to 3.3 min for the zebrafish dataset and from 48 600 to 8 min for the CeNGEN dataset, respectively (Fig. 1, available as supplementary data at *Bioinformatics* online). On the zebrafish dataset, FastSCODE achieved its maximum performance across all computing systems when utilizing two or three GPUs (Fig. 1B). However, on the CeNGEN dataset, FastSCODE achieved its maximum performance across most computing systems when utilizing three or four GPUs (Fig. 1C).

FastSCODE achieved speed improvements of up to 505-fold on the mESC dataset and 1050-fold on the skin cancer dataset when utilizing the NVIDIA RTX 4090 GPU and PRO 6000, respectively (Fig. 2, available as supplementary data at *Bioinformatics* online). Both datasets were small enough for FastSCODE to compute the entire dataset on a single GPU at once. However, despite increasing the number of GPU devices, the total execution time unexpectedly increased (Figs 1 and 2, available as supplementary data at *Bioinformatics* online).

3.2 Resource utilization analysis

Since increasing the number of GPU devices did not necessarily result in performance improvement, we measured and analyzed both the time spent on computing tasks and the temporal variation of CPU and GPU utilization (%) during runtime (Figs 3–5, available as supplementary data at *Bioinformatics* online). These results reveal a distinct trade-off between CPU-to-GPU memory transfer and GPU computation. As the number of devices increases, the proportion of time spent on CPU-to-GPU data transfer grows markedly, indicating a substantial communication overhead introduced by multi-device parallelization.

Consequently, effective multi-GPU acceleration can be achieved only when the computational workload is sufficiently large to dominate the data-transfer overhead, as observed in large-scale datasets such as zebrafish and CeNGEN.

3.3 Detailed performance analysis

We examined how the batch size, one of the most critical hyperparameters, affects the runtime performance of FastSCODE (Figs 6 and 7, available as supplementary data at *Bioinformatics* online). When the batch size is set to the entire dataset, the overhead from data transfer was reduced, leading to a substantial enhancement in computation speed. Increasing the batch size for matrix **B** while reducing the number of optimization iterations decreased execution time, highlighting the importance of reducing repeated sampling and evaluation steps in the optimization process. We also compared manycore acceleration frameworks (Figs 8 and 9, available as supplementary data at *Bioinformatics* online). For larger datasets, parallelization effectively amortizes computational overhead across GPUs, narrowing the performance differences among frameworks. In contrast, for smaller datasets, data transfer overhead dominates and amplifies performance gaps, particularly in frameworks showing lower parallel efficiency. The consistency between the results of FastSCODE and the original SCODE was also evaluated (Fig. 10, available as supplementary data at *Bioinformatics* online).

4 Conclusion

FastSCODE is an accelerated implementation of SCODE that uses parallel processing on manycore processors to enhance computational efficiency. FastSCODE was designed to accelerate SCODE by introducing batch array computing into the linear regression phase, targeting the elimination of repetitive computations for each gene. Our experiments demonstrate that FastSCODE achieves substantial scalability and runtime improvements in the GRN inference from large-scale scRNA-seq datasets. FastSCODE eliminates the major computational bottlenecks of SCODE, transforming it into a highly scalable and efficient algorithm. Given its performance and scalability, FastSCODE offers a practical and efficient solution for GRN inference in bioinformatics and biomedical research.

Author contributions

Rakbin Sung (Conceptualization [equal], Data curation [equal], Formal analysis [equal], Investigation [equal], Methodology [equal], Resources [equal], Software [equal], Validation [equal], Visualization [equal], Writing—original draft [equal], Writing—review & editing [equal]), Seongmi Woo (Data curation [equal], Investigation [equal], Methodology [equal], Resources [equal], Software [equal], Validation [equal], Writing—original draft [equal]), Dongmin Shin (Data curation [equal], Formal analysis [equal], Investigation [equal], Methodology [equal], Software [equal], Validation [equal], Writing—original draft [equal]), Junil Kim (Conceptualization [equal], Funding acquisition [equal], Supervision [equal], Validation [equal], Writing—original draft [equal]), and Daewon Lee (Conceptualization [equal], Investigation [equal], Methodology [equal], Project administration [equal], Supervision [equal], Writing—original draft [equal], Writing—review & editing [equal])

Supplementary data

Supplementary data is available at *Bioinformatics* online.

Conflict of interest: None declared.

Funding

This work was supported by the Chung-Ang University Graduate Research Scholarship in 2024. This work was also supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) [RS-2025-02263724, RS-2024-00342721].

Data availability

FastSCODE with test dataset is available online for public use at https://github.com/cxinsys/fastscode

References

Abadi M, Agarwal A, Barham P *et al.* TensorFlow: large-scale machine learning on heterogeneous systems. 2015. Software available from tensorflow.org

Andrews TS, Kiselev VY, McCarthy D *et al.* Tutorial: guidelines for the computational analysis of single-cell RNA sequencing data. *Nat Protoc* 2021;**16**:1–9.

Bradbury J, Frostig R, Hawkins P *et al. JAX: Composable Transformations of Python+NumPy Programs*. 2018. Software available from https://github.com/jax-ml/jax

Chen L, Dautle M, Gao R *et al.* Inferring gene regulatory networks from time-series scrna-seq data via granger causal recurrent autoencoders. *Brief Bioinform* 2025;**26**:bbaf089.

Denyer T, Ma X, Klesen S *et al.* Spatiotemporal developmental trajectories in the Arabidopsis root revealed using high-throughput single-cell RNA sequencing. *Dev Cell* 2019;**48**:840–52.e5.

Eisenstein M. Single-cell RNA-seq analysis software providers scramble to offer solutions. *Nat Biotechnol* 2020;**38**:254–7.

Gaydosik AM, Queen DS, Trager MH *et al.* Genome-wide transcriptome analysis of the STAT6-regulated genes in advanced-stage cutaneous T-cell lymphoma. *Blood J Am Soc Hematol* 2020;**136**:1748–59.

Granja JM, Corces MR, Pierce SE *et al.* ArchR is a scalable software package for integrative single-cell chromatin accessibility analysis. *Nat Genet* 2021;**53**:403–11.

Hammarlund M, Hobert O, Miller DM *et al.* The cengen project: the complete gene expression map of an entire nervous system. *Neuron* 2018;**99**:430–3.

Harris CR, Millman KJ, van der Walt SJ *et al.* Array programming with NumPy. *Nature* 2020;**585**:357–62.

Ji AL, Rubin AJ, Thrane K *et al.* Multimodal analysis of composition and spatial architecture in human squamous cell carcinoma. *Cell* 2020;**182**:497–514.e22.

Kfoury Y, Baryawno N, Severe N *et al.*; as part of the Boston Bone Metastases Consortium. Human prostate cancer bone metastases have an actionable immunosuppressive microenvironment. *Cancer Cell* 2021;**39**:1464–78.e8.

Kim H, Choi H, Lee D *et al.* A review on gene regulatory network reconstruction algorithms based on single cell RNA sequencing. *Genes Genomics* 2024;**46**:1–11.

Li X, Li B, Gu S *et al.* Single-cell and spatial RNA sequencing reveal the spatiotemporal trajectories of fruit senescence. *Nat Commun* 2024;**15**:3108.

Matsumoto H, Kiryu H, Furusawa C *et al.* SCODE: an efficient regulatory network inference algorithm from single-cell RNA-Seq during differentiation. *Bioinformatics* 2017;**33**:2314–21.

Moerman T, Aibar Santos S, Bravo González-Blas C *et al.* GRNBoost2 and arboreto: efficient and scalable inference of gene regulatory networks. *Bioinformatics* 2019;**35**:2159–61.

Okuta R, Unno Y, Nishino D *et al.* CuPy: A NumPy-compatible library for NVIDIA GPU calculations. In: *Proceedings of the 31st Conference on Neural Information Processing Systems*, 151(7). Long Beach, CA, 2017.

Paszke A, Gross S, Massa F *et al.* PyTorch: an imperative style, high- performance deep learning library. In: *Advances in Neural Information Processing Systems 32*. Vancouver, BC, Canada, 2019, p. 8024–35.

Pratapa A, Jalihal AP, Law JN *et al.* Benchmarking algorithms for gene regulatory network inference from single-cell transcriptomic data. *Nat Methods* 2020;**17**:147–54.

Ramachandran P, Matchett KP, Dobie R *et al.* Single-cell technologies in hepatology: new insights into liver biology and disease pathogenesis. *Nat Rev Gastroenterol Hepatol* 2020;**17**:457–72.

Setty M, Tadmor MD, Reich-Zeliger S *et al.* Wishbone identifies bifurcating developmental trajectories from single-cell data. *Nat Biotechnol* 2016;**34**:637–45.

Shu H, Zhou J, Lian Q *et al.* Modeling gene regulatory networks using neural network architectures. *Nat Comput Sci* 2021;**1**: 491–501.

Su M, Pan T, Chen Q-Z *et al.* Data analysis guidelines for single-cell RNA-seq in biomedical studies and clinical applications. *Mil Med Res* 2022;**9**:68.

Sung R, Kim H, Kim J *et al.* FastTENET: an accelerated TENET algorithm based on manycore computing in Python. *Bioinformatics* 2024;**40**:btae699.

Tuck AC, Natarajan KN, Rice GM *et al.* Distinctive features of lincrna gene expression suggest widespread RNA-independent functions. *Life Sci Alliance* 2018;**1**:e201800124.

Wagner DE, Weinreb C, Collins ZM *et al.* Single-cell mapping of gene expression landscapes and lineage in the zebrafish embryo. *Science* 2018;**360**:981–7.

Wang M, Song W-M, Ming C *et al.* Guidelines for bioinformatics of single-cell sequencing data analysis in Alzheimer's disease: review, recommendation, implementation and application. *Mol Neurodegener* 2022;**17**:17.

Yossef R, Krishna S, Sindiri S *et al.* Phenotypic signatures of circulating neoantigen-reactive CD8+ T cells in patients with metastatic cancers. *Cancer Cell* 2023;**41**:2154–65.e5.

Zhang T-Q, Chen Y, Wang J-W *et al.* A single-cell analysis of the Arabidopsis vegetative shoot apex. *Dev Cell* 2021;**56**:1056–74.e8.

Zinati Y, Takiddeen A, Emad A *et al.* GRouNdGAN: GRN-guided simulation of single-cell RNA-seq data using causal generative adversarial networks. *Nat Commun* 2024;**15**:4055.

Bioinformatics, 2025, 41, 1–5
https://doi.org/10.1093/bioinformatics/btaf624
Applications Note