scDataset: Scalable Data Loading for Deep Learning on Large-Scale Single-Cell Omics

Davide D'Ascenzo [1 2] **Sebastiano Cultrera di Montesano** [3]

Abstract

Modern single-cell datasets now comprise hundreds of millions of cells, presenting significant challenges for training deep learning models that require shuffled, memory-efficient data loading. While the AnnData format is the community standard for storing single-cell datasets, existing data loading solutions for AnnData are often inadequate: some require loading all data into memory, others convert to dense formats that increase storage demands, and many are hampered by slow random disk access. We present scDataset, a PyTorch IterableDataset that operates directly on one or more AnnData files without the need for format conversion. The core innovation is a combination of block sampling and batched fetching, which together balance randomness and I/O efficiency. On the Tahoe 100M dataset, scDataset achieves up to a 48× speed-up over AnnLoader, a 27× speed-up over HuggingFace Datasets, and an 18× speed-up over BioNeMo in single-core settings. These advances democratize large-scale single-cell model training for the broader research community.

1. Introduction

The advent of single-cell omics has transformed molecular biology by enabling the high-throughput measurement of gene expression, chromatin accessibility, and protein abundance at the resolution of individual cells. Among these, single-cell transcriptomics has emerged as the most widely adopted modality, providing genome-wide snapshots of gene expression that reveal cellular identity, state, and function. These technologies have driven major discoveries across cancer biology (Tirosh et al., 2016), neurodevelopment (Nowakowski et al., 2017), and immunology (Villani et al., 2017), uncovering rare cell types, lineage hierarchies, and dynamic transcriptional programs (Wang & Navin, 2015; Buenrostro et al., 2015; Regev et al., 2017).

As the scale and accessibility of single-cell experiments continue to grow, public datasets have expanded rapidly. As of October 2024, CELLxGENE Discover hosts over 1,550 datasets and more than 93 million unique cells, spanning a wide range of tissues, diseases, and experimental conditions (CZI Cell Science Program et al., 2024). This resource is primarily observational, capturing the natural diversity of cell states across biological contexts. More recently, the Tahoe 100M dataset marked a new milestone for interventional single-cell datasets, combining both scale and experimental richness: it profiles over 100 million cells across 379 drug perturbations, 47 cancer cell lines, and multiple dosages—totaling more than 17,000 perturbation contexts (Zhang et al., 2025). Together, these large-scale resources provide a foundation for modeling cell states and biological responses at unprecedented resolution.

To extract structure from these datasets, the field is increasingly turning to deep learning models that can learn expressive representations of cellular states from raw omics data. Recent efforts have focused on building biological foundation models—large, pre-trained architectures designed to generalize across tissues, conditions, and experimental settings (Bommasani et al., 2021). Models such as GeneFormer (Theodoris et al., 2023) and scGPT (Cui et al., 2024) leverage transformer-based architectures to capture gene-gene dependencies, and can be fine-tuned for tasks such as cell type classification, batch correction, and perturbation prediction. This direction has created strong demand for scalable infrastructure to support model training on datasets containing hundreds of millions of cells, with the long-term goal of building in silico "virtual cells" that simulate cellular behavior across modalities and contexts (Bunne et al., 2024).

The AnnData format has become the community standard for storing single-cell omics data, offering efficient handling of sparse matrices, metadata, and annotations (Virshup et al., 2024). However, its integration with deep learning work-

[1]Department of Computer Science, University of Milan, Milan, Italy [2]Department of Control and Computer Engineering, Politecnico di Torino, Torino, Italy [3]Eric and Wendy Schmidt Center, Broad Institute of MIT and Harvard, Cambridge, MA, USA. Correspondence to: Davide D'Ascenzo <davide.dascenzo@unimi.it>, Sebastiano Cultrera di Montesano <scultrer@broadinstitute.org>.

Proceedings of the 42nd International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

flows remains underdeveloped. Standard training algorithms such as stochastic gradient descent (SGD) require diverse, randomly sampled minibatches to ensure stable convergence and generalization (Bottou, 1999; Keskar et al., 2016). Ensuring this diversity at hundred-million-cell scale typically requires loading the full dataset into memory, which is infeasible for most practitioners. Alternative strategies, such as converting to dense formats or relying on random disk access, suffer from inflated storage costs or low throughput. For instance, AnnLoader—an experimental PyTorch data loader developed by the AnnData team to enable on-disk sampling—achieves only 20 samples/second on the Tahoe 100M dataset, requiring *more than 58 days* for a single training epoch.

We introduce scDataset, a PyTorch IterableDataset designed for efficient and scalable training on large single-cell omics datasets. scDataset operates directly on one or more AnnData files—without format conversion or full in-memory loading—and implements a quasi-random sampling strategy that combines block sampling with batched fetching. Block sampling reduces the number of random disk reads by accessing contiguous chunks, while batched fetching amortizes I/O latency and enables in-memory reshuffling for minibatch diversity. While scDataset was developed to address the limitations of AnnData in deep learning applications, its design is broadly applicable to any large-scale dataset stored on disk. In benchmarking on the Tahoe 100M dataset, scDataset achieved up to a 48× speed-up over AnnLoader. Furthermore, as AnnLoader does not natively support multiprocessing, we evaluated scDataset in a multiprocessing configuration and observed a 129× speed-up compared to AnnLoader in single-core mode. We also conducted systematic benchmarks against HuggingFace Datasets (Lhoest et al., 2021), a widely used dataset library, and BioNeMo-SCDL (John et al., 2024), a specialized single-cell data loading solution developed by NVIDIA. scDataset improved the performance of both backends, achieving a 27× speed-up over HuggingFace Datasets and an 18× speed-up over BioNeMo.

The source code and documentation for scDataset are publicly available on GitHub at `https://github.com/Kidara/scDataset`.

Current solutions The AnnData format has become the community standard for single-cell omics data, underpinning widely used analysis tools such as Scanpy (Wolf et al., 2018). Its popularity stems from flexible support for sparse matrices, rich metadata, and seamless integration with the Python data science ecosystem. While AnnData supports on-disk access via backing modes, most deep learning tools—including frameworks like scvi-tools (Gayoso et al., 2021)—often require loading the entire dataset into memory. As modern single-cell atlases span hundreds of millions of cells and multiple terabytes of storage, this is increasingly infeasible. The only native option, **AnnLoader**, preserves format compatibility but suffers from slow throughput and lacks multiprocessing support, making it impractical for large-scale use.

In response, various groups have developed custom data loading pipelines tailored for deep learning. These typically convert AnnData to alternative formats, introducing new technical and interoperability challenges. For example, GeneFormer (Theodoris et al., 2023) converts data to **HuggingFace Datasets**, while scTab (Fischer et al., 2024) transforms AnnData into Parquet files for use with NVIDIA Merlin. These conversions often require densifying sparse matrices, significantly increasing storage requirements. Other groups have developed custom storage infrastructures, such as the Zarr-based backend in SCimilarity (Heimberg et al., 2025), or adopted entirely new formats like TileDB's SOMA in CZ CELLxGENE (CZI Cell Science Program et al., 2024). A notable tailored solution is **BioNeMo-SCDL**, developed by NVIDIA to support large-scale single-cell training (John et al., 2024). It introduces a custom format and leverages NumPy memory mapping for efficient loading of the `X` matrix. However, it still requires conversion from AnnData and does not natively handle the full metadata stack, limiting compatibility with the broader analysis ecosystem.

In summary, while AnnData remains central to the field and does support on-disk access in principle, no efficient, standard solution exists for deep learning workflows. Existing alternatives either sacrifice compatibility and flexibility or fall short on performance. Our work addresses this gap by introducing a faster and more flexible solution that operates directly on AnnData files—eliminating format conversion and enabling scalable model training on modern single-cell datasets.

2. Method

scDataset is designed to enable efficient, randomized data loading from large-scale datasets. It serves as a flexible and extensible interface that connects diverse data backends, such as AnnData and HuggingFace Datasets, to PyTorch's DataLoader. The core technical innovations of scDataset are its *block sampling* and *batched fetching* strategies, which together balance I/O efficiency with minibatch diversity. To accommodate a wide range of data sources and workflows, scDataset is implemented as a PyTorch IterableDataset with a modular architecture. The design centers on four user-configurable functions that govern data retrieval and transformation, enabling seamless integration with arbitrary data backends (see Figure 1).

Appendix A.1 discusses the rationale for adopting an

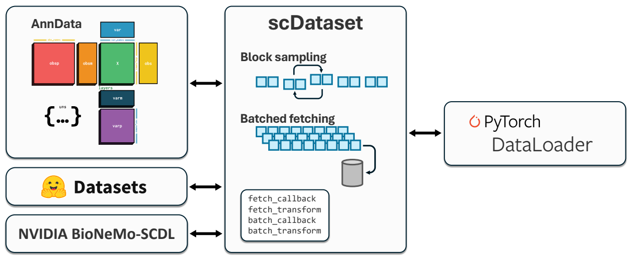

Figure 1. scDataset bridges diverse data backends with PyTorch's DataLoader through a modular interface. Data retrieval is managed by a configurable `fetch_callback`, followed by preprocessing with `fetch_transform` (e.g., sparse-to-dense conversion). Batches are selected using `batch_callback` and further processed with `batch_transform` before being yielded to the training pipeline.

iterable-style dataset over the traditional map-style paradigm in PyTorch. Detailed descriptions of the block sampling and batched fetching strategies are provided in Appendix A.2 and Appendix A.3, respectively.

Evaluation setup We benchmarked scDataset on the Tahoe 100M dataset (Zhang et al., 2025), which is available in three formats: AnnData (14 files of approximately 7 million cells each), HuggingFace Datasets, and BioNeMo. The AnnData files (downloadable from the official GitHub[1]) occupy 314GB on disk. The HuggingFace version[2] requires 1.9TB, and the BioNeMo format, generated using the official conversion script[3], occupies 1.1TB, with peak storage usage of 2.2TB during conversion.

For the HuggingFace dataset, a custom script was used to load the `X` matrix. The BioNeMo dataset stores only the `X` matrix, so handling metadata requires additional custom logic. When applied to AnnData, scDataset yields batches as AnnData objects, preserving all original metadata. In contrast, when used with HuggingFace or BioNeMo, scDataset currently returns only the `X` matrix, as metadata must be managed separately.

All experiments were conducted on an NVIDIA DGX Station with 256GB RAM, an Intel Xeon E5-2698 v4 CPU, and 5TB of solid-state drive (SSD) storage. Unless otherwise specified, a fixed batch size of 64 was used for all benchmarks.

Data loading throughput We evaluated data loading speed by measuring single-core throughput (samples per second). Each data loader was warmed up for 30 seconds, followed by 120 seconds of measurement per condition. We benchmarked scDataset using seven block sizes (1, 2, 4, 8, 16, 32, 64) and seven fetch factors (1, 2, 4, 8, 16, 32, 64) on the AnnData, HuggingFace, and BioNeMo datasets. For AnnData, AnnLoader served as a baseline.

The performance of scDataset depends not only on the sampling strategy but also on how efficiently the storage backend serves data requests. While scDataset delivers requests in an optimal format for batch retrieval, the actual data access pattern—whether batched or as individual queries—is determined by the backend implementation. As a result, improvements from increasing the fetch factor are observed only when the backend supports efficient batched reads. Notably, very large fetch factors can slightly degrade throughput due to the increased computational overhead of in-memory shuffling of large buffers.

Results are presented in Figure 2 (AnnData), Figure 3 (HuggingFace), and Figure 4 (BioNeMo). As expected, scDataset with block size 1 and fetch factor 1 matches the baseline performance. On AnnData, increasing both block size and fetch factor yields up to a **48**× speed-up over AnnLoader (block size 64, fetch factor 64). For HuggingFace and BioNeMo, throughput improves with larger block sizes, reaching up to **27**× and **18**× speed-ups, respectively, at block size 64.

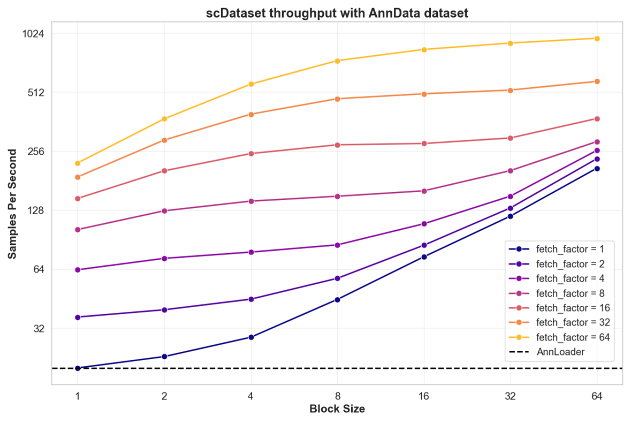

Figure 2. Data loading throughput for scDataset on the AnnData dataset as a function of block size and fetch factor. Throughput (samples/sec) increases substantially with larger block sizes and higher fetch factors, demonstrating that both parameters synergistically improve I/O efficiency. At the largest tested values, scDataset achieves over 48× higher throughput compared to the baseline.

Minibatch diversity We assessed sampling quality by measuring plate label entropy within minibatches. In the Tahoe 100M dataset, each of the 14 AnnData files corresponds to a unique plate label, and because the dataset is concatenated without prior shuffling, adjacent cells share the same label.

We evaluated scDataset using the same seven block sizes and fetch factors as in the throughput benchmarks. Since minibatch diversity is determined solely by the sampling

[1]https://github.com/ArcInstitute/arc-virtual-cell-atlas
[2]https://huggingface.co/datasets/tahoebio/Tahoe-100M
[3]https://nvidia.github.io/bionemo-framework/API_reference/bionemo/scdl/scripts/convert_h5ad_to_scdl/

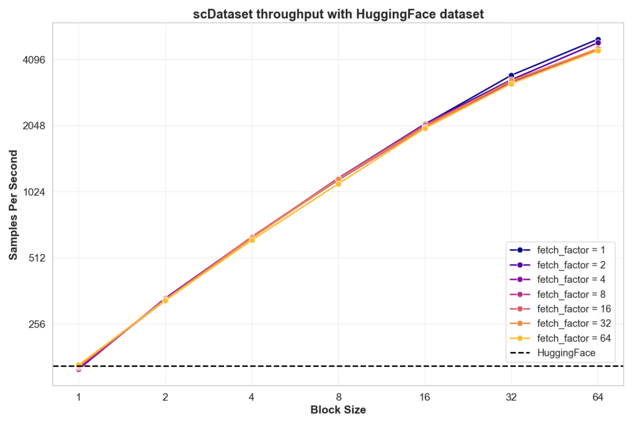

Figure 3. Data loading throughput for scDataset on the HuggingFace dataset as a function of block size and fetch factor. Throughput increases with larger block sizes, but remains unaffected by the fetch factor. At the largest block size, scDataset achieves a 27× speed-up over the baseline.

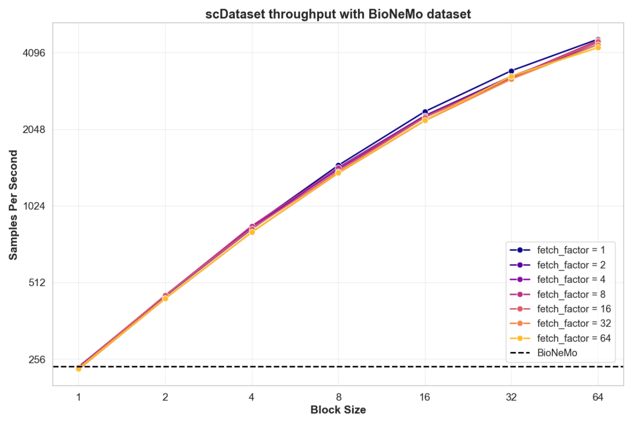

Figure 4. Data loading throughput for scDataset on the BioNeMo dataset as a function of block size and fetch factor. Throughput increases with larger block sizes, but remains unaffected by the fetch factor. At the largest block size, scDataset achieves an 18× speed-up over the baseline.

strategy, this analysis was performed once, independent of backend. Both AnnLoader and scDataset with block size 1 and fetch factor 1 achieve random shuffling, yielding an entropy of approximately 3.63. As shown in Figure 5, increasing block size reduces entropy—reaching zero at block size 64—while higher fetch factors counteract this effect and help maintain diversity.

Scaling throughput with multiprocessing We evaluated the maximum throughput of scDataset on the AnnData dataset using multiprocessing, a feature not natively supported by AnnLoader. The hyperparameter search space is summarized in Table 1, with full results in Table 2. While no single configuration is universally optimal, we highlight a setting with `block_size`=4, `fetch_factor`=16, and `num_workers`=12, which achieves approximately 2593

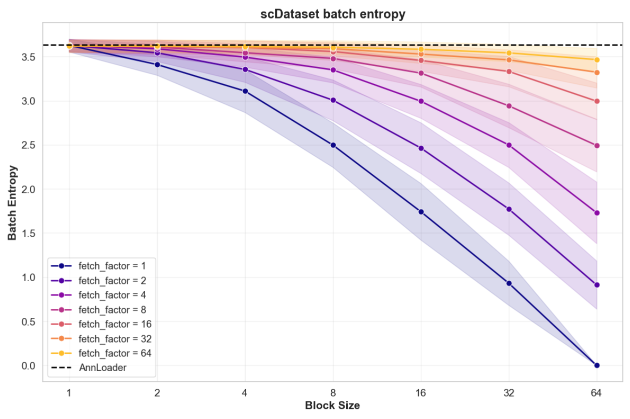

Figure 5. Plate label entropy within minibatches for scDataset as a function of block size and fetch factor. Higher entropy indicates greater minibatch diversity.

samples/sec and maintains an entropy of 3.59—comparable to random sampling. This represents a **129**× speed-up over AnnLoader. Notably, training for one epoch on the full Tahoe 100M dataset would take **over 58 days** with AnnLoader, but **less than 11 hours** with scDataset.

Table 1. Hyperparameter search space for throughput experiments with multiprocessing on the AnnData dataset.

Parameter	**Values**
Block size (b)	4, 8, 16, 32
Fetch factor (f)	4, 8, 16, 32
Number of workers	8, 12, 16

3. Discussion

This work introduces **scDataset**, a scalable and flexible data loader for training deep learning models on large-scale single-cell omics datasets. By combining block sampling and batched fetching, scDataset enables randomized, high-throughput training directly from disk without requiring format conversion or full in-memory loading. The implementation integrates directly with PyTorch, provides native support for multiprocessing, and consistently delivers substantial speed-ups over existing solutions such as AnnLoader, HuggingFace Datasets, and BioNeMo. By operating directly on formats like AnnData, scDataset enables shuffled training on commodity hardware and lowers the barrier to large-scale deep learning in single-cell biology.

While scDataset delivers strong practical performance, two limitations remain. First, our assessment of sampling quality is based on plate label entropy—a metadata-based measure that captures known batch structure but may overlook more subtle correlations or biases. Second, the current implementation does not support weighted sampling, which is important for handling imbalanced datasets or enabling stratified sampling of rare subpopulations. Future work will focus on incorporating additional metrics for sampling quality and extending support to weighted and stratified sampling—improving the flexibility and applicability of scDataset across increasingly diverse biological settings.

Acknowledgements

Davide D'Ascenzo was financially supported by the Italian National PhD Program in Artificial Intelligence (DM 351 intervento M4C1 - Inv. 4.1 - Ricerca PNRR), funded by NextGenerationEU (EU-NGEU). Sebastiano Cultrera di Montesano was supported by the Eric and Wendy Schmidt Center at the Broad Institute of MIT and Harvard.

Impact Statement

This work helps democratize large-scale model training to the broader scientific community.

References

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., Brynjolfsson, E., Buch, S., Card, D., Castellon, R., Chatterji, N. S., Chen, A. S., Creel, K. A., Davis, J., Demszky, D., Donahue, C., Doumbouya, M. K. B., Durmus, E., Ermon, S., Etchemendy, J., Ethayarajh, K., Fei-Fei, L., Finn, C., Gale, T., Gillespie, L., Goel, K., Goodman, N. D., Grossman, S., Guha, N., Hashimoto, T., Henderson, P., Hewitt, J., Ho, D. E., Hong, J., Hsu, K., Huang, J., Icard, T. F., Jain, S., Jurafsky, D., Kalluri, P., Karamcheti, S., Keeling, G., Khani, F., Khattab, O., Koh, P. W., Krass, M. S., Krishna, R., Kuditipudi, R., Kumar, A., Ladhak, F., Lee, M., Lee, T., Leskovec, J., Levent, I., Li, X. L., Li, X., Ma, T., Malik, A., Manning, C. D., Mirchandani, S., Mitchell, E., Munyikwa, Z., Nair, S., Narayan, A., Narayanan, D., Newman, B., Nie, A., Niebles, J. C., Nilforoshan, H., Nyarko, J. F., Ogut, G., Orr, L. J., Papadimitriou, I., Park, J. S., Piech, C., Portelance, E., Potts, C., Raghunathan, A., Reich, R., Ren, H., Rong, F., Roohani, Y. H., Ruiz, C., Ryan, J., R'e, C., Sadigh, D., Sagawa, S., Santhanam, K., Shih, A., Srinivasan, K. P., Tamkin, A., Taori, R., Thomas, A. W., Tramèr, F., Wang, R. E., Wang, W., Wu, B., Wu, J., Wu, Y., Xie, S. M., Yasunaga, M., You, J., Zaharia, M. A., Zhang, M., Zhang, T., Zhang, X., Zhang, Y., Zheng, L., Zhou, K., and Liang, P. On the opportunities and risks of foundation models. *ArXiv*, abs/2108.07258, 2021.

Bottou, L. *On-line learning and stochastic approximations*, pp. 9–42. Cambridge University Press, USA, 1999. ISBN 0521652634.

Buenrostro, J. D., Wu, B., Litzenburger, U., Ruff, D. W., Gonzales, M. L., Snyder, M. P., Chang, H. Y., and Greenleaf, W. J. Single-cell chromatin accessibility reveals principles of regulatory variation. *Nature*, 523:486 – 490, 2015.

Bunne, C., Roohani, Y., Rosen, Y., Gupta, A., Zhang, X., Roed, M., Alexandrov, T., AlQuraishi, M., Brennan, P., Burkhardt, D. B., Califano, A., Cool, J., Dernburg, A. F., Ewing, K., Fox, E. B., Haury, M., Herr, A. E., Horvitz, E., Hsu, P. D., Jain, V., Johnson, G. R., Kalil, T., Kelley, D. R., Kelley, S. O., Kreshuk, A., Mitchison, T., Otte, S., Shendure, J., Sofroniew, N. J., Theis, F., Theodoris, C. V., Upadhyayula, S., Valer, M., Wang, B., Xing, E., Yeung-Levy, S., Zitnik, M., Karaletsos, T., Regev, A., Lundberg, E., Leskovec, J., and Quake, S. R. How to build the virtual cell with artificial intelligence: Priorities and opportunities. *Cell*, 187(25):7045–7063, Dec 2024. ISSN 0092-8674. doi: 10.1016/j.cell.2024.11.015.

Cui, H., Wang, C., Maan, H., Pang, K., Luo, F., Duan, N., and Wang, B. scgpt: toward building a foundation model for single-cell multi-omics using generative ai. *Nature Methods*, 21(8):1470–1480, Aug 2024. ISSN 1548-7105. doi: 10.1038/s41592-024-02201-0.

CZI Cell Science Program, Abdulla, S., Aevermann, B., Assis, P., Badajoz, S., Bell, S. M., Bezzi, E., Cakir, B., Chaffer, J., Chambers, S., Cherry, J., Chi, T., Chien, J., Dorman, L., Garcia-Nieto, P., Gloria, N., Hastie, M., Hegeman, D., Hilton, J., Huang, T., Infeld, A., Istrate, A.-M., Jelic, I., Katsuya, K., Kim, Y. J., Liang, K., Lin, M., Lombardo, M., Marshall, B., Martin, B., McDade, F., Megill, C., Patel, N., Predeus, A., Raymor, B., Robatmili, B., Rogers, D., Rutherford, E., Sadgat, D., Shin, A., Small, C., Smith, T., Sridharan, P., Tarashansky, A., Tavares, N., Thomas, H., Tolopko, A., Urisko, M., Yan, J., Yeretssian, G., Zamanian, J., Mani, A., Cool, J., and Carr, A. Cz cellxgene discover: a single-cell data platform for scalable exploration, analysis and modeling of aggregated data. *Nucleic Acids Research*, 53(D1):D886–D900, 11 2024. ISSN 1362-4962. doi: 10.1093/nar/gkae1142.

Fischer, F., Fischer, D. S., Mukhin, R., Isaev, A., Biederstedt, E., Villani, A.-C., and Theis, F. J. sctab: Scaling cross-tissue single-cell annotation models. *Nature Communications*, 15(1):6611, Aug 2024. ISSN 2041-1723. doi: 10.1038/s41467-024-51059-5.

Gayoso, A., Lopez, R., Xing, G., Boyeau, P., Wu, K., Jayasuriya, M., Melhman, E., Langevin, M., Liu, Y., Samaran, J., Misrachi, G., Nazaret, A., Clivio, O., Xu, C., Ashuach, T., Lotfollahi, M., Svensson, V., Beltrame, E. d. V., Talavera-López, C., Pachter, L., Theis, F. J., Streets, A., Jordan, M. I., Regier, J., and Yosef, N. scvi-tools: a library for deep probabilistic analysis of single-cell omics data. *bioRxiv*, 2021. doi: 10.1101/2021.04.28.441833.

Heimberg, G., Kuo, T., DePianto, D. J., Salem, O., Heigl, T., Diamant, N., Scalia, G., Biancalani, T., Turley, S. J., Rock, J. R., Corrada Bravo, H., Kaminker, J., Vander Heiden, J. A., and Regev, A. A cell atlas foundation model for scalable search of similar human cells. *Nature*, 638 (8052):1085–1094, Feb 2025. ISSN 1476-4687. doi: 10.1038/s41586-024-08411-y.

John, P. S., Lin, D., Binder, P., Greaves, M., Shah, V., John, J. S., Lange, A., Hsu, P. D., Illango, R., Ramanathan, A., Anandkumar, A., Brookes, D. H., Busia, A., Mahajan, A., Malina, S., Prasad, N., Sinai, S., Edwards, L., Gaudelet, T., Regep, C., Steinegger, M., Rost, B., Brace, A., Hippe, K., Naef, L., Kamata, K., Armstrong, G., Boyd, K., Cao, Z., Chou, H.-Y., Chu, S., dos Santos Costa, A., Darabi, S., Dawson, E., Didi, K., Fu, C., Geiger, M., Gill, M., Hsu, D. J., Kaushik, G., Korshunova, M., Kothen-Hill, S. T., Lee, Y., Liu, M., Livne, M., McClure, Z., Mitchell, J., Moradzadeh, A., Mosafi, O., Nashed, Y. L., Paliwal, S., Peng, Y., Rabhi, S., Ramezanghorbani, F., Reidenbach, D., Ricketts, C., Roland, B., Shah, K., Shimko, T., Sirelkhatim, H., Srinivasan, S., Stern, A. C., Toczydlowska, D., Veccham, S. P., Venanzi, N. A. E., Vorontsov, A., Wilber, J., Wilkinson, I., Wong, W. J., Xue, E., Ye, C., Yu, X., Zhang, Y., Zhou, G., Zandstein, B., Dallago, C., Trentini, B., Kucukbenli, E., Rvachov, T., Calleja, E., Israeli, J., Clifford, H., Haukioja, R., Haemel, N., Tretina, K., Tadimeti, N., and Costa, A. B. Bionemo framework: a modular, high-performance library for ai model development in drug discovery. *ArXiv*, abs/2411.10548, 2024.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. *ArXiv*, abs/1609.04836, 2016.

Lhoest, Q., del Moral, A. V., Jernite, Y., Thakur, A., von Platen, P., Patil, S., Chaumond, J., Drame, M., Plu, J., Tunstall, L., Davison, J., vSavsko, M., Chhablani, G., Malik, B., Brandeis, S., Scao, T. L., Sanh, V., Xu, C., Patry, N., McMillan-Major, A., Schmid, P., Gugger, S., Delangue, C., Matussiere, T., Debut, L., Bekman, S., Cistac, P., Goehringer, T., Mustar, V., Lagunas, F., Rush, A. M., and Wolf, T. Datasets: A community library for natural language processing. *ArXiv*, abs/2109.02846, 2021.

Nowakowski, T. J., Bhaduri, A., Pollen, A. A., Alvarado, B., Mostajo-Radji, M. A., Lullo, E. D., Haeussler, M., Sandoval-Espinosa, C., Liu, S. J., Velmeshev, D., Ounadjela, J. R., Shuga, J., Wang, X., Lim, D. A., West, J. A. A., Leyrat, A. A., Kent, W. J., and Kriegstein, A. R. Spatiotemporal gene expression trajectories reveal developmental hierarchies of the human cortex. *Science*, 358: 1318 – 1323, 2017.

Regev, A., Teichmann, S. A., Lander, E. S., Amit, I., Benoist, C., Birney, E., Bodenmiller, B., Campbell, P., Carninci, P., Clatworthy, M. R., Clevers, H., Deplancke, B., Dunham, I., Eberwine, J., Eils, R., Enard, W., Farmer, A., Fugger, L., Göttgens, B., Hacohen, N., Haniffa, M. A., Hemberg, M., Kim, S., Klenerman, P., Kriegstein, A. R., Lein, E. S., Linnarsson, S., Lundberg, E., Lundeberg, J., Majumder, P. P., Marioni, J. C., Merad, M., Mhlanga, M. M., Nawijn, M. C., Netea, M., Nolan, G. P., Pe'er, D., Phillipakis, A., Ponting, C. P., Quake, S. R., Reik, W., Rozenblatt-Rosen, O., Sanes, J. R., Satija, R., Schumacher, T. N., Shalek, A. N., Shapiro, E. Y., Sharma, P., Shin, J. W., Stegle, O., Stratton, M., Stubbington, M. J. T., Theis, F. J., Uhlen, M., van Oudenaarden, A., Wagner, A., Watt, F. M., Weissman, J. S., Wold, B., Xavier, R., and Yosef, N. The human cell atlas. *eLife*, 6, 2017.

Theodoris, C. V., Xiao, L., Chopra, A., Chaffin, M. D., Al Sayed, Z. R., Hill, M. C., Mantineo, H., Brydon, E. M., Zeng, Z., Liu, X. S., and Ellinor, P. T. Transfer learning enables predictions in network biology. *Nature*, 618(7965):616–624, Jun 2023. ISSN 1476-4687. doi: 10.1038/s41586-023-06139-9.

Tirosh, I., Izar, B., Prakadan, S. M., Wadsworth, M. H., Treacy, D. J., Trombetta, J. J., Rotem, A., Rodman, C., Lian, C. G., Murphy, G., Fallahi-Sichani, M., Dutton-Regester, K., Lin, J.-R., Cohen, O., Shah, P., Lu, D., Genshaft, A. S., Hughes, T., Ziegler, C. G. K., Kazer, S. W., Gaillard, A., Kolb, K. E., Villani, A.-C., Johannessen, C. M., Andreev, A., Allen, E. M. V., Bertagnolli, M. M., Sorger, P. K., Sullivan, R. J., Flaherty, K. T., Frederick, D. T., Jané-Valbuena, J., Yoon, C. H., Rozenblatt-Rosen, O., Shalek, A. K., Regev, A., and Garraway, L. A. Dissecting the multicellular ecosystem of metastatic melanoma by single-cell rna-seq. *Science*, 352:189 – 196, 2016.

Villani, A.-C., Satija, R., Reynolds, G., Sarkizova, S., Shekhar, K., Fletcher, J., Griesbeck, M., Butler, A., Zheng, S., Lazo, S., Jardine, L., Dixon, D., Stephenson, E., Nilsson, E., Grundberg, I., McDonald, D., Filby, A., Li, W., Jager, P. L. D., Rozenblatt-Rosen, O., Lane, A. A., Haniffa, M. A., Regev, A., and Hacohen, N. Single-cell rna-seq reveals new types of human blood dendritic cells, monocytes, and progenitors. *Science*, 356, 2017.

Virshup, I., Rybakov, S., Theis, F. J., Angerer, P., and Wolf, F. A. anndata: Access and store annotated data matrices. *Journal of Open Source Software*, 9(101):4371, 2024. doi: 10.21105/joss.04371.

Wang, Y. X. and Navin, N. E. Advances and applications of single-cell sequencing technologies. *Molecular cell*, 58 4:598–609, 2015.

Wolf, F. A., Angerer, P., and Theis, F. J. Scanpy: large-scale single-cell gene expression data analysis. *Genome*

Biology, 19(1):15, Feb 2018. ISSN 1474-760X. doi: 10.1186/s13059-017-1382-0.

Zhang, J., Ubas, A. A., de Borja, R., Svensson, V., Thomas, N., Thakar, N., Lai, I., Winters, A., Khan, U., Jones, M. G., Tran, V., Pangallo, J., Papalexi, E., Sapre, A., Nguyen, H., Sanderson, O., Nigos, M., Kaplan, O., Schroeder, S., Hariadi, B., Marrujo, S., Salvino, C. C. A., Gallareta Olivares, G., Koehler, R., Geiss, G., Rosenberg, A., Roco, C., Merico, D., Alidoust, N., Goodarzi, H., and Yu, J. Tahoe-100m: A giga-scale single-cell perturbation atlas for context-dependent gene function and cellular modeling. *bioRxiv*, 2025. doi: 10.1101/2025.02.20.639398.

A. Appendix

A.1. Map-style vs iterable-style PyTorch Datasets

PyTorch supports two primary paradigms for dataset implementation: *map-style* and *iterable-style* datasets.

Map-style datasets implement the `__getitem__` method, allowing retrieval of individual samples by index. Batching is managed by the DataLoader, which assembles minibatches by collecting samples one at a time and merging them via a `collate_fn`. This approach is efficient when the dataset resides in memory, as random access is fast. However, for on-disk datasets, this results in a large number of random I/O operations per minibatch—a significant bottleneck, especially on hard disk drives (HDDs). For instance, SCimilarity employs a map-style dataset, which constrains disk throughput (Heimberg et al., 2025).

An exception is the experimental AnnLoader and its underlying AnnCollection dataset, which extend `__getitem__` to accept both single indices and batches of indices. By providing a `batch_sampler` to the DataLoader's `sampler` argument, minibatches can be retrieved in a single call, reducing I/O overhead. While this approach deviates from standard PyTorch API usage, it does improve throughput by minimizing disk accesses.

Iterable-style datasets, in contrast, require implementation of the `__iter__` method, returning an iterator over samples. This paradigm offers maximal flexibility, enabling custom sampling and loading strategies. However, it precludes the use of standard PyTorch samplers, which rely on map-style indexing, and necessitates careful handling of multiprocessing within the iterable implementation.

Given the limitations of map-style datasets for large, on-disk single-cell data, we adopt an iterable-style dataset for scDataset. This design empowers us to implement efficient sampling strategies optimized for disk-based access patterns. Our implementation natively supports reading from single or multiple AnnData files without format conversions and integrates custom multiprocessing logic aligned with PyTorch's DataLoader specifications.

A.2. Block sampling

Training deep learning models relies on stochastic gradient descent (SGD), where the diversity of each minibatch is crucial for unbiased updates and robust convergence (Bottou, 1999). Achieving this diversity via random sampling is straightforward in-memory, but becomes challenging when datasets are too large to fit in memory and must be accessed from disk.

Disk drives, particularly HDDs, are optimized for reading large contiguous chunks of data, but are inefficient when required to perform frequent, small, non-sequential reads. While SSDs mitigate this limitation to some extent, contiguous access remains significantly faster than random access.

To reconcile the need for random sampling with the realities of disk I/O, we introduce a *block sampling* strategy (see Algorithm 1). Instead of sampling each data point independently, we randomly select contiguous blocks of size b (e.g., 8 cells) from the dataset. For a minibatch size m (e.g., $m = 256$) and block size $b = 8$, only $m/b = 32$ random disk reads are required per minibatch, each retrieving a contiguous chunk of b samples. These blocks are then assembled to form the final minibatch.

The choice of block size b governs the trade-off between I/O efficiency and minibatch diversity. Larger blocks improve throughput by reducing the number of random disk operations, but may group together cells with correlated metadata (e.g., from the same tissue or batch), potentially reducing diversity. Conversely, smaller blocks enhance randomness but increase I/O overhead. This parameter can be tuned based on dataset properties and hardware capabilities.

A.3. Batched fetching

While block sampling reduces the number of random disk reads, further improvements in throughput and randomness can be achieved through *batched fetching*. In this approach, multiple blocks are prefetched into memory in a single I/O operation, forming a buffer that is subsequently reshuffled to construct diverse minibatches.

Batched fetching amortizes the latency of disk access across larger data transfers, which is particularly beneficial on high-latency storage devices. After fetching a batch of blocks, the samples are randomly permuted in memory before being yielded as minibatches, ensuring that each minibatch contains a diverse set of cells.

The detailed procedure for batched fetching is presented in **Algorithm 2**. This algorithm outlines how scDataset preloads

Algorithm 1 Block Sampling

```
Input  : Dataset size $n$, block size $b$, minibatch size $m$
         (where $n$ is a multiple of $b$ and $m$ for simplicity)
Output : Sequence of minibatches $\mathcal{M}_0, \mathcal{M}_1, \ldots$
1  Generate full index array: $I = [0, 1, \ldots, n-1]$
2  Split $I$ into $k = \frac{n}{b}$ blocks $[B_0, B_1, \ldots, B_{k-1}]$,
     where $B_i = [i \cdot b, \ldots, (i+1) \cdot b - 1]$
3  Shuffle block order:
     $[B_{\sigma(0)}, \ldots, B_{\sigma(k-1)}] \leftarrow \text{RandomPerm}([B_0, \ldots, B_{k-1}])$
4  Concatenate shuffled blocks:
     $I_{\text{shuffled}} \leftarrow B_{\sigma(0)} \| \ldots \| B_{\sigma(k-1)}$
5  Split $I_{\text{shuffled}}$ into minibatches $[M_0, \ldots, M_{\frac{n}{m}-1}]$ of size $m$
6  for each $M_i$ do
7      Load data: $\mathcal{M}_i \leftarrow \text{ReadFromDisk}(M_i)$
8      yield $\mathcal{M}_i$
```

blocks, reshuffles their contents, and yields randomized minibatches, balancing I/O efficiency with quasi-random sampling.

Algorithm 2 Block Sampling with Batched Fetching

```
Input  : Dataset size $n$, block size $b$, minibatch size $m$, fetch factor $f$
         (where $n$ is a multiple of $b$ and $m \cdot f$ for simplicity)
Output : Sequence of minibatches $\mathcal{M}_0, \mathcal{M}_1, \ldots$
1  Generate full index array: $I = [0, 1, \ldots, n-1]$
2  Split $I$ into $k = \frac{n}{b}$ blocks $[B_0, B_1, \ldots, B_{k-1}]$,
     where $B_i = [i \cdot b, \ldots, (i+1) \cdot b - 1]$
3  Shuffle block order:
     $[B_{\sigma(0)}, \ldots, B_{\sigma(k-1)}] \leftarrow \text{RandomPerm}([B_0, \ldots, B_{k-1}])$
4  Concatenate shuffled blocks:
     $I_{\text{shuffled}} \leftarrow B_{\sigma(0)} \| \ldots \| B_{\sigma(k-1)}$
5  Split $I_{\text{shuffled}}$ into batches $[F_0, \ldots, F_{\frac{n}{m \cdot f}-1}]$ of size $m \cdot f$
6  for each $F_i$ do
7      Load data: $\mathcal{F}_i \leftarrow \text{ReadFromDisk}(F_i)$
8      Shuffle $\mathcal{F}_i$ in memory
9      Split $\mathcal{F}_i$ into minibatches $\mathcal{M}_0, \ldots, \mathcal{M}_j$
10     for each $\mathcal{M}_j$ do
11         yield $\mathcal{M}_j$
```

Table 2. Results for throughput experiments with multiprocessing on the AnnData dataset. The experiment highlighted in **bold** corresponds to the configuration referenced in the main text.

Block size	Fetch factor	Num workers	Samples/sec	Avg. batch entropy	Std. batch entropy
4	4	8	597	3.51	0.11
		12	882	3.49	0.11
		16	1175	3.50	0.11
	8	8	1061	3.56	0.10
		12	1626	3.57	0.09
		16	2131	3.56	0.09
	16	8	1876	3.59	0.08
		12	**2593**	**3.59**	**0.09**
		16	2492	3.59	0.08
	32	8	1852	3.60	0.08
		12	1768	3.61	0.08
		16	1761	3.61	0.08
8	4	8	666	3.34	0.15
		12	993	3.34	0.16
		16	1323	3.33	0.16
	8	8	1181	3.48	0.12
		12	1766	3.49	0.12
		16	2297	3.48	0.12
	16	8	2093	3.55	0.09
		12	2603	3.56	0.10
		16	2529	3.55	0.10
	32	8	1873	3.59	0.08
		12	1774	3.59	0.09
		16	1789	3.59	0.08
16	4	8	861	3.00	0.23
		12	1308	3.00	0.23
		16	1697	3.01	0.22
	8	8	1319	3.33	0.16
		12	1940	3.32	0.16
		16	2573	3.32	0.16
	16	8	2266	3.48	0.12
		12	2612	3.48	0.12
		16	2549	3.47	0.12
	32	8	1878	3.54	0.10
		12	1834	3.55	0.10
		16	1775	3.54	0.10
32	4	8	1309	2.47	0.27
		12	1932	2.47	0.27
		16	2510	2.47	0.28
	8	8	1662	3.00	0.23
		12	2493	3.00	0.22
		16	3122	3.00	0.22
	16	8	2474	3.32	0.16
		12	2570	3.32	0.16
		16	2458	3.31	0.16
	32	8	1837	3.48	0.12
		12	1740	3.46	0.13
		16	1781	3.46	0.12