PharmaHacks 2024 Participant Handbook

PharmaHacks Machine Learning team

Contents

1) Introduction

1.a) About this guide

Welcome to PharmaHacks! We're thrilled that you have chosen to participate. This guide is meant to help you familiarize yourself with key bioinformatics concepts to keep in mind during the hackathon. PharmaHacks welcomes people who come from biology as well as from a quantitative/CS background. Regardless of your background, we like to think that you will find it useful. We advise all participants to do a read-through but feel free to skip parts that you are already comfortable with.

We tried to give an overview of some key concepts in machine learning and in the two subfields covered by our challenges: **neural data analysis** and **computational genomics**. On the weekend of the challenge, you will have the choice between two challenges, one for each of these subfields. You will be making use of ML/data science concepts for both.

If you don't do anything else, **please go through the Environment Setup** at the end of this guide.

1.b) General approach to the hackathon

Bioinformatics is all about applying techniques from statistics, Machine Learning (ML), and high-performance computing to concrete medical and scientific problems. To do this effectively, one must be able to bridge those fields with a combination of pragmatism and understanding.

This is a delicate line to walk, as there is significant complexity beneath both the data analysis methods being used and the biological questions that are being asked. Therefore, **one must make sure to pinpoint the aspects of the problem at hand that are most relevant to the creation of a solution, and discard the rest.**

While this may sound daunting, we assure you that, to solve bioinformatics problems, you do not necessarily need to be an expert in biology, nor do you need to be an expert computer scientist. **Asking the right questions** and having an **eye for what matters** is often more important than having expert knowledge.

When solving the challenge of your choice, remember that **there is no singular 'right' answer**. Rather, there is a large breadth of different approaches you can take to solving the challenges, drawing from your creativity and scientific thinking. You cannot do everything - so **focus on what you think matters most**! You may also favor what you feel able to do well over more advanced approaches – or treat the event primarily as a learning experience and shoot for the moon (who knows, it might just work!). Either approach can make strategic sense depending on your and your team's situation.

If you feel overwhelmed by the amount of information that you have to deal with and feel out of your depth, don't hesitate to **ask your teammates**! Teaming up with people with skills that are complementary to your own is a powerful strategy. You will also have the option to ask questions during the hackathon.

If you have access to GPT-4 through ChatGPT Plus (or a comparable model, such as Claude 3 Opus), you may find it very helpful - not necessarily for advanced questions, but most definitely to understand the basics. If you choose to use it to solve the challenge, that is fine – especially for specific coding questions – but please ensure that you are using it as a tool and **understanding what you are doing** rather than simply regurgitating its outputs.

2) General ML techniques and tools

2.a) You're given a dataset. What do you do?

This simple question is perhaps one of the most important to answer. On the day of the hackathon, you will choose a challenge pertaining to neural data analysis or genomics. You will be given a problem statement and a dataset or a collection of datasets. Where to go from here?

First, you will have to **load it**, from persistent storage to memory. Some data formats are easier to work with than others, but libraries are generally available to let you load data with minimal effort. If your dataset is large, as is often the case with biological data, you might need to be careful about how you load it in order not to run out of memory. This is usually done by only loading what you need on demand, instead of loading everything at once.

After that, while you might be tempted to immediately reach for advanced algorithms, the most important thing to do is to **make sense of the data**. This is where *domain knowledge* - knowledge that is specific to the field or problem at hand - starts to become important. How to make sense of data depends on what you are working with, and often starts with simple statistics or visualizations. For example, if you are given a set of entries of the form $(\text{experiment index}, x, y, z, \text{time})$ representing trajectories, you could choose entries corresponding to a few experiments, and display their trajectories in an interactive 3D plot, to get a feel for what they look like.

In Python, most data scientists and computational biologists will use a tool like the **pandas** library to wrangle their data. **Polars** is also a great choice, but it is newer and changing quickly, so you might have a harder time finding tutorials. These libraries will let you easily transform and analyze your data, and are a **must-have** in your computational toolbox. If you haven't used them yet, do not panic: this is an opportunity to do so! There are plenty of learning resources online, such as the Kaggle[1] pandas course, which belongs to a wider set of data science courses that you may find helpful.

To **visualize your data**, matplotlib is most commonly used. Seaborn is built upon it, provides some extra features and plot types, and is meant to work closely with Pandas.

Once you've loaded your data and understood its structure and basic properties (which falls in the scope of Exploratory Data Analysis), your next task will often be to start **building a predictive model** that leverages it.

2.b) Building Machine Learning models

2.b.a) Key references

If you haven't done any Machine Learning before, you may want to take a look at the Kaggle "Intro to Machine Learning" tutorial, in order to get the lay of the land. If you don't have prior experience, try to get a feel of how things fit together, rather than going deep into the underlying theory. **Big picture first!**

At a minimum, you should familiarize yourself with the following concepts, using online resources of your choice:

- Basic algorithms such as **linear/logistic regression**
- **Dimensionality reduction**, notably **Principal Component Analysis (PCA)**, **t-SNE** and **UMAP**
- What metrics to use, for which problem: **Mean Squared Error (MSE)**, **F1 score**, **R2**…
- Cross-validation

[1]Kaggle is a popular machine learning and data science platform that offers a variety of challenges and learning resources.

2.b.b) Algorithm selection and feature engineering

You should know that you can get surprisingly far with simple models. Linear/logistic regression, for example, is one of the simplest algorithms you can use, but it can give great results if your data is appropriately structured. Sometimes, that means transforming your data so that those algorithms are easier to apply to it. This idea is called **feature engineering**. If you're not familiar with it, Kaggle has a series of tutorials on the topic, which we highly recommend taking a look at.

2.b.c) The case of deep learning

We feel that Artificial Neural Networks (ANNs – what **deep learning** is all about) deserve a special mention. Nowadays, when people refer to "AI," they often mean "deep learning." Deep learning is tremendously powerful; it powers essentially all modern speech recognition software (including ChatGPT, Google/Bing Search, etc.), and many more.

Deep neural networks trained on large amounts of data typically need much less feature engineering, if any, than other classes of models. In general, the simpler a model is, the likelier it is to benefit from careful feature engineering. You can think of it as shifting the burden of understanding some properties of the data between the model, and the modeler (you).

This sounds great, but it comes with caveats. The most relevant to you in the context of this challenge are **data hunger** and **computational requirements.**

If you don't have much data, it may simply be unfeasible to train a large neural network, as it will **overfit**. In this case, you may consider using a smaller network, applying *regularization* or *data augmentation*, or even moving away from deep learning and using a more traditional class of algorithms.

You may also be unable to train a sufficiently large network because of computational limitations: deep learning should always be done on GPUs (typically NVIDIA, because of software compatibility issues) or TPUs, and you may not have enough time and power to train a network for your use case. Exercise caution.

If you decide to use deep learning and do not have sufficiently powerful hardware on your compute, we recommend using Google Colab (more info in Section 5.b), which offers access to free cloud GPUs.

2.b.d) Foundation models

In recent years, the concept of **foundation models** [1] has been making waves. They are now a major focus of investment and research in a variety of fields. This concept is specific to deep learning.

The "usual" way to create deep learning models is to train a model on a dataset specific to the problem at hand, typically in a supervised fashion. For example, if you have a dataset of 1000 pictures of dogs and 1000 pictures of cats, you could train a Convolutional Neural Network model to distinguish cats from dogs, by starting from scratch and using just these images.

In a foundation model mindset, instead of starting from scratch, you're starting from an existing model that has been trained on much, much more data than what you have at hand. It is not rare for such models to be trained on hundreds of millions, billions, or even trillions of data points. This process, called **pretraining**, is expensive and time-consuming, but once it is done, the result can be specialized (**fine-tuned**) for a variety of tasks in a very efficient manner. If the annotated dataset you're using is small, you might get much better performance by leveraging a foundation model than by trying to solely work with your own data. This has classically been known as **transfer learning**, but nomenclature has shifted towards *foundation models* with the advent of Large Language Models with so-called emergent abilities, particularly GPT-3 and its successors.

Why do foundation models work? At a very high and intuitive level, the idea is that by being exposed to huge amounts of data, they have "seen the world" a lot, and learned about its structure in general, repurposable ways. It is possible to train them on such vast amounts of data because they typically require either no or little annotation. In other words, if you have 100 million images, you can feed them to a foundation model for pretraining without paying human beings to sit down and annotate each image to ascertain what's on it. You can just remove parts of those images randomly, and ask the model to reconstruct the missing parts using the ones that are still there. In doing so, the model will have to understand the structure of those images: if it sees a dog's body and the head is missing, and it can infer the dog's breed from seeing its body, then it can make a decent guess at what the dog's head might look like. This structural understanding can then be applied to a variety of tasks in an efficient manner.

So far, we've been talking about foundation models in the context of image processing. They're also very prevalent in Natural Language Processing (you have certainly heard of GPT-3 and GPT-4), and are gaining traction in life sciences [2], with projects such as Deep Genomics' BigRNA, Evo, scBERT [3], scGPT [4], and more.

By now, you may agree that foundation models are pretty exciting, but they come with a few caveats:

- They are essentially all **large deep learning models**, which you cannot run at an acceptable speed without a **GPU that has enough VRAM** (how much VRAM depends on the specific model).
- **Fine-tuning** a foundation model requires more VRAM than just running it, although this can be alleviated by "freezing" most of it, or using techniques such as LoRA.
- You must make sure that the data the foundation model has been trained on exhibits **sufficient similarity** with the one you want to perform fine-tuning on.
- **A foundation model is nothing without its weights**. If a paper describes a foundation model but you cannot download its checkpoints/weights (binary files of a few tens of MB to tens of GB), you cannot do anything with it, even if the code is provided. No matter what, **you are not going to train your own foundation model here**. So, before you invest significant time into a given model, make sure you can actually use it!
- Foundation models are great, but **they aren't magic**: they don't remove the need for careful thinking when it comes to your general approach, data preprocessing or correctness.

2.b.e) Explainability / interpretability

Explainability is always a great property in a model. It is the ability to tell why a model is acting a certain way, what it is basing its predictions on. Without explainability, AI models are so-called *black boxes*: data comes in, answers come out… but why?

Simple models are typically more explainable than complex ones such as deep neural networks. There are strategies to make the internals of deep learning models clearer, but they tend to be highly problem-specific or computationally intensive.

If you have sufficiently few input features (columns) and they are individually meaningful (as in, you can put words on what each feature represents), you may be able to use Shapley scores. They are used to determine the importance of each feature by quantifying its contribution to the prediction. This helps in understanding how different features influence the model's output, making the model's decisions more transparent and interpretable. A popular library to do this in Python is **shap**, which offers an introduction to both its usage and the concept of Shapley values.

2.c) Ensuring correctness

So you have a model, and it gives pretty good results. You're super happy! It's almost too good to be true. Perhaps it is, unfortunately.

Throughout the development of any ML model, you **must keep an eye out for correctness**. This is of vital importance, as not doing so can compromise your entire solution, invalidating all of your results. Correctness, or conceptual soundness, is about **making sure that your model isn't cheating**.

For example, if you train a tumor detection model (which is a predictive model) on a set of images, and then evaluate it on those same images, you **cannot trust the results** that you are getting, because your model may be overfitting to those images. This is an important point, as it does not apply to the way most *scientific* models have historically been developed (which largely still applies, outside of ML-based methods). It is crucial that you understand the concepts of training, validation and test data sets, and keep them in mind as you develop your solution. For PharmaHacks, you don't have to separate the validation set and the test set - having a separate training and testing set will be enough.

A model that gives great performance, but only on the data that it has been trained on, is not actually a good model.

Even if you take care to separate your data into training and testing sets and do cross-validation (which you should), your model might still be unsound! When the model is accidentally exposed to data that it shouldn't have seen, that is data leakage. Instances of data leakage can range from nearly obvious to very subtle - in the latter case, even experienced researchers can sometimes miss them. As you build your model, try to think if there's anything in the way you're splitting your data, which might leak too much information between training and testing.

2.d) Justifying your choices

ML can sometimes be something of an art. There is often a good explanation for why a given approach works better than another. However, that is not always the case; such an explanation often is often anchored in intuition rather than in theoretical proofs. When there is a proof, it is often order of magnitude harder to understand than it is to "make things work."

We want to stress that **it's OK not to have definite answers**, Trying things out is part of the process, and, ultimately, the performance of your model (as evaluated through appropriate metrics for the problem at hand) is going to be of crucial importance. You might not know why a learning rate of 6×10^{-3} worked better than 5×10^{-3}, and that's fine.

That being said, being able to **justify your choices** pertaining to key model properties such as data representation, preprocessing, training objective, etc., will be valuable. The academics on the jury will be happy that you have approached problems with a scientific mindset, rather than simply throwing things at the wall and seeing what sticks.

2.e) Additional resources

These are not essential but may be of interest.

- Feature selection: most relevant for simple models and high-level features.
- Ensemble learning: combining models to get something greater than the sum of their parts.

3) Neural decoding

3.a) Overview

One of the challenges will deal with **neural decoding**.

Key idea: The brain of an organism must be encoding what that organism is perceiving, intending to do, or doing, and that this data can be decoded from the activity of its neurons. In other words, by looking at how and when specific neurons spike, you might be able to tell where a mouse thinks it is, how it is about to move its foot, or what it's smelling.

At a very basic level, this can be done using purely statistical methods: you might find direct correlations between the activities of single neurons and some behaviorally relevant variable. As it turns out, single neurons can be surprisingly expressive, and something as basic as plotting their average firing rate with respect to various experimental conditions (such as whether the subject is being shown a face or an object) can already reveal a great deal of information. Such an approach does not require machine learning in its most-commonly accepted forms, just linear/logistic regression.

This, however, has its limitations. While single neurons are informative, it is more and more widely accepted that they are not enough to get the full picture, and that *groups*, or "populations," of neurons encode important information in their collective activity patterns. Dimensionality reduction techniques (e.g. PCA, t-SNE, UMAP) can reveal low-dimensional structure in high-dimensional neural population activity. Clustering methods can identify groups of neurons with similar response properties. ML models can be trained to decode stimuli, behaviors, or cognitive states from neural population activity.

Neural population decoding is an active area of research. New state-of-the-art methods are appearing at a fast rate. **We found CEBRA [5] and POYO-1 [6] of particular interest, and encourage you to take a look at them** if you are interested in neural decoding. In order to meaningfully visualize data, you may also find rastermap [7] of interest.

3.b) Imaging modalities

There are various common imaging techniques, or "modalities," used to record neural activity that later becomes the data to analyze. Different recording modalities have different strengths and limitations, so it's important to pay attention to which one is being used.

- **Calcium imaging**: Calcium is an essential compound for neural information transmission. Thus, measuring changes in calcium levels, with calcium indicators, both in neurons and cells that help neurons (glia) can provide important insights into neural activity. This method offers high spatial resolution (at cellular and subcellular levels), and the same neurons can be repeatedly imaged over time. However, it has relatively low temporal resolution (often tens or hundreds of milliseconds), and requires much post-processing.
- **Functional Magnetic Resonance Imaging (fMRI)**: measures the small changes in blood flow occurring during neural activity, treating them as correlates of neural activity. It offers spatial resolution in the millimeter range, but low temporal resolution (seconds).
- **Electroencephalography (EEG)**: Electroencephalographic recordings measure brain activity patterns through scalp electrodes and are used in many contexts (e.g., diagnosis of epileptic seizures, monitoring anesthesia depth, etc.) due to their ease of deployment, non-invasive nature and high sampling rate (hundreds to thousands of hertz). However, EEG recordings are limited to superficial brain regions and have poor spatial resolution.
- **Magnetoencephalography (MEG)**: MEG recordings map neural activity by measuring the magnetic fields generated by the electrical currents produced naturally by neurons. Benefits of MEG recordings include strong temporal and spatial resolution (in the millimeters and milliseconds) and its non-invasive nature. Source localization remains challenging, with low signal-to-noise ratios for activity in deeper brain regions.
- **Single-neuron electrophysiological recordings**: directly measuring the electrical activity of individual neurons. This technique is highly invasive, as it requires the placement of microelectrodes near or into the neuron(s) of interest, but provides excellent spatial and temporal resolution. Note that you can obtain single-neuron recordings from many neurons simultaneously, each recording forming a separate stream of data. Nevertheless, the recorded populations tend to be smaller than with other methods.

3.c) Questions to ask about neural data

When analyzing neural data, it will be important to consider:

- What type of recording is it (e.g. calcium imaging, electrophysiology, EEG, fMRI)?
- What is the spatial resolution (e.g. single cells, cortical columns, brain regions) and temporal resolution (e.g. milliseconds, seconds)?
- How many neurons/channels/voxels were recorded simultaneously? More neurons means more potential for understanding population codes.
- What regions are being recorded from? What is known about their function?

For the neural decoding challenge, **you will be working with calcium imaging data**, which will be available in both in its raw form and in pre-processed (deconvolved) form.

3.d) Loading large neural datasets efficiently

Lazy loading is a programming technique based on deferring the loading of non-essential resources until they are actually needed. This can be essential with large datasets – as is common when working with neural data – since they might not fit into your computer or server's RAM. This is even more important in case the data does not even fit on the long-term storage (HDD, SSD) of your machine.

For the neural challenge, you will be working with NWB/HDF5 data, briefly described below.

Hierarchical Data Format Version 5 (HDF5):

- File format that organizes data hierarchically, using a "files and folders"-like structure. This is useful since neural data is commonly divided into internal groups like acquisition, processing, analysis and stimuli depending on the type of data.
- Commonly used for high-performance scientific computing.
- Particularly useful for lazy-loading. Lazy loading is natively supported in its standard Python bindings, hdf5py.

NWB (Neurodata Without Borders):

- Standard for organizing neural data, built upon HDF5.
- NWB natively supports lazy loading as it uses HDF5.
- Dandi Archive: repository, with plenty of NWB files, designed to facilitate open-access sharing of neural data.

3.e) Getting started

We mentioned CEBRA earlier in this guide. We highly recommend checking out the paper [5], as well as the documentation of the official Python package. In particular, we think the following guides will be helpful:

- Using CEBRA-Time to Analyze OpenScope Data: working with NWB data
- Decoding from a CEBRA embedding: doing neural decoding using CEBRA

Obviously, CEBRA is not the only way to decode neural data (we also mentioned POYO-1 [6] previously). There are many other ways to do so. However, it is a modern method, exhibiting good performance, with easy-to-use Python bindings and clear guides to get started with it.

4) Analyzing scRNA-seq data

Another challenge will deal with **computational genomics**.

Almost all of our cells contain the same genetic material (i.e., DNA), but not all DNA is transcribed to RNA in every cell, and this process depends on many factors such as cell type, developmental stage, environmental conditions, and the presence of specific signaling molecules, among many others.

These factors are crucial in determining the cell's function and its role in health and disease. Understanding how and why different genes are expressed in different cells helps us unravel the complex mechanisms of diseases, offering insights into their diagnosis, progression, and treatment. **Single-cell RNA sequencing (scRNA-seq)** is a powerful tool that measures the transcriptomes of individual cells to give a high-resolution cell view of transcript abundance. This technology has immense power in revealing cellular heterogeneity, enabling researchers to pinpoint specific cell types and states associated with disease processes and therapeutic responses and also can be used to associate gene expression patterns between conditions.

However, most studies to date have focused on identifying differentially expressed genes between cell types [8], and differential abundance of cells between states [9], [10]. This might have been because this technology was more expensive in its early years, but its cost is decreasing exponentially [11], leading to an increase in the number of available cells and patients in studies. Thus, with this increase in the volume of data, the next milestone would be to use this high-resolution scRNA-seq data to build predictive models that can predict patient outcomes while shedding light on the possible underlying mechanisms of diseases.

However, we get sparse matrices of gene expression data from single-cell RNA sequencing experiments, with a very high number of columns. In this regime, the curse of dimensionality is a concern. It is practically impossible to use that data for prediction purposes (or any other purpose) without transforming it first to get fewer, more informative features.

UMAP is frequently used in single-cell genomics for the efficient identification of cell types by transforming high-dimensional data to a lower-dimensional space, preserving both local and global data structures to highlight cellular heterogeneity. You can read more about it **here**. While very useful and widely used, keep in mind that **UMAP is a means to an end**. It is a way of dealing with the extremely high dimensionality of gene expression data, and bringing it down to a number of dimensions that are tractable for analysis and visualization.

As you wait for the event, we encourage you to consider the challenges involved in leveraging scRNA-seq data.

You may find the following resources useful:

- **Introduction to single-cell RNA-Seq and Seurat**
- **Reading single-cell data in R**
- **MIT Deep Learning Genomics - Lecture 11 - RNA, PCA, t-SNE, Embeddings**
- **Documentation for Seurat**, which is an important toolkit for handling RNA-seq data
- **Tutorial on performing basic analysis using Seurat objects**

5) Environment setup

To make sure that you don't waste precious time on the weekend of the challenge, we recommend setting up your working environment beforehand.

We recommend using Python for the neural decoding challenge. For the genomics challenge, whether to use R or Python is up to you. You can even use both, provided that you understand how to export data from one and load it from the other. If you wish to use deep learning, Python will be practically unavoidable, even in the genomics challenge.

5.a) System preparation

- Ensure that you have sufficient disk space: we recommend a minimum of a few tens of GB. If you choose to pursue the neural decoding challenge, this may go up to 300GB if you opt not to use lazy loading and to run things locally.
- If you have already used R or Python, make sure your installation is sufficiently up-to-date and that you are still able to install the package and run the example code.

5.b) Python on Google Colab

If your laptop lacks a powerful gaming GPU or you lack confidence in your Python installation, we recommend using Google Colab to run your experiments/training. It offers free access to GPUs in an interactive Jupyter environment.

- Go to the **Colab** homepage and make sure that you can create, save and load a new notebook. We recommend going through the "Welcome to Colab" tutorial notebook provided by Google.
- Try to use a GPU or a TPU as a "Hardware accelerator." **This will be crucial if you want to run deep-learning models**. You can do that by clicking **"Runtime" > "Change runtime type"**.
- Make sure that you can install dependencies: try running `!pip install numpy` (mind the exclamation mark!) in a new cell. It should print "Requirement already satisfied."
- You may find **this tutorial** helpful.

Note: a standard Colab runtime should offer 12 GB of CPU RAM. This is not to be confused with GPU or TPU RAM, which will be most important when running deep learning models

5.c) R/RStudio locally

You will most likely want to use R to solve the Genomics challenge. If you are interested in it, you should install it on your own computer. R kernels are available on Colab, but installing dependencies can take an excessive amount of time, so it is safer to run things locally.

- Install R and RStudio **here**.
- For Mac users: you will need to install **XQuartz** and **gfortran**
- Install the `devtools` and `Seurat` packages by running this block of code in RStudio:

```
install.packages("devtools")
library(devtools)
install.packages("Seurat")
library(Seurat)
```

Note: Installing these packages may take around 10-15 minutes if they need to be compiled on your machine.

6) Conclusion

If you've made it this far, congratulations! We hope this background guide will help you throughout the exciting weekend to come. Good luck and happy hacking!

— ML Team, PharmaHacks 2024

References

[1] R. Bommasani *et al.*, “On the Opportunities and Risks of Foundation Models”. 2022.

[2] J. Ma and B. Wang, “Towards foundation models of biological image segmentation”, *Nature Methods*, vol. 20, no. 7, pp. 953–955, Jul. 2023, doi: 10.1038/s41592-023-01885-0.

[3] F. Yang *et al.*, “scBERT as a large-scale pretrained deep language model for cell type annotation of single-cell RNA-seq data”, *Nature Machine Intelligence*, vol. 4, no. 10, pp. 852–866, Oct. 2022, doi: 10.1038/s42256-022-00534-z.

[4] H. Cui *et al.*, “scGPT: toward building a foundation model for single-cell multi-omics using generative AI”, *Nature Methods*, Feb. 2024, doi: 10.1038/s41592-024-02201-0.

[5] S. Schneider, J. H. Lee, and M. W. Mathis, “Learnable latent embeddings for joint behavioural and neural analysis”, *Nature*, vol. 617, no. 7960, pp. 360–368, May 2023, doi: 10.1038/s41586-023-06031-6.

[6] M. Azabou *et al.*, “A Unified, Scalable Framework for Neural Population Decoding”. 2023.

[7] C. Stringer, L. Zhong, A. Syeda, F. Du, and M. Pachitariu, “Rastermap: a discovery method for neural population recordings”, *bioRxiv*, 2023, doi: 10.1101/2023.07.25.550571.

[8] A. L. Thurman, J. A. Ratcliff, M. S. Chimenti, and A. A. Pezzulo, “Differential gene expression analysis for multi-subject single-cell RNA-sequencing studies with aggregateBioVar”, *Bioinformatics*, vol. 37, no. 19, pp. 3243–3251, 2021, doi: 10.1093/bioinformatics/btab337.

[9] N. Millard *et al.*, “Maximizing statistical power to detect differentially abundant cell states with scPOST”, *Cell Reports Methods*, vol. 1, no. 8, p. 100120, 2021, doi: https://doi.org/10.1016/j.crmeth.2021.100120.

[10] J. Zhao *et al.*, “Detection of differentially abundant cell subpopulations in scRNA-seq data”, *Proceedings of the National Academy of Science*, vol. 118, no. 22, p. e2100293118, Jun. 2021, doi: 10.1073/pnas.2100293118.

[11] B. Hwang, J. H. Lee, and D. Bang, “Single-cell RNA sequencing technologies and bioinformaticspipelines”, *Experimental & Molecular Medicine*, vol. 50, no. 8, pp. 1–14, Aug. 2018, doi: 10.1038/s12276-018-0071-8.